THE DEFINITIVE GUIDE TO

Machine Learning Platforms

By Sam Charrington
Founder, TWIML
Host, TWIML AI Podcast

twiml

Table of Contents

Introduction

In spite of all the hype, the reality is that you won't need to do anything different for your company to benefit from AI. You won't need to hire data scientists, you won't need to collect any training data, and you won't need to build any machine learning or deep learning models.

It is already the case, and it will be increasingly true in the future, that AI technologies will power many of the products and services that your enterprise uses. Your enterprise will be a passive beneficiary of AI.

These passive benefits, however, are a rising tide that floats every boat. They are undifferentiated—your enterprise gets only what is available to everyone else in the marketplace—and, as a result, they are also undifferentiating. In other words, they won't help your enterprise gain a competitive advantage among its peers. So, the question becomes: what *will* help your enterprise gain a competitive advantage?

Competing on Models

Achieving competitive advantage with AI requires a much more active approach. It is done by applying your enterprise's **proprietary data** to solve your enterprise's **proprietary business problems** through the creation of **proprietary models**. (When we refer to "Enterprise AI" in this ebook, we're referring to this idea.) The benefits of such an approach can be significant.

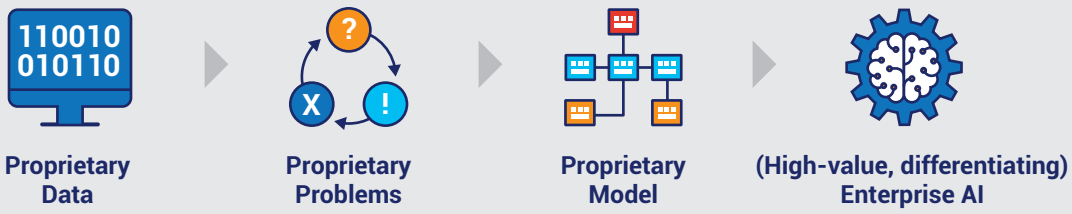

Figure 1. Enterprise AI: Using proprietary data to create proprietary models that solve proprietary problems.

In a 2016 paper on The Netflix Recommender System, authors Neil Hunt (former chief product officer) and Carlos Gomez-Uribe (former VP of product innovation), reported: "over years of development of personalization and recommendations, we have reduced churn by several percentage points. Reduction of monthly churn both increases the lifetime value of an existing subscriber, and reduces the number of new subscribers we need to acquire to replace canceled members." The authors estimated that "the combined effect of personalization and recommendations save[d Netflix] more than $1B per year."[1]

Another example of the advantage gained through proprietary models comes from a 2018 TWIML interview[2] with Jeff Dean, the head of Google AI. In explaining the impact of the company's investment in machine learning models, Jeff cited the replacement of the complex, decades-old, phrase-based translation system used to power Google Translate with a system based on deep learning. Not only is the resulting system much simpler—a system of around 500,000 lines of code was replaced with around 500 lines of TensorFlow code—but the performance gains in making this change exceeded the performance gains of the previous decades of work on the legacy system.

With examples like these, it's no surprise that enterprises across a wide variety of industries are investing significantly in machine learning.

Enter the Model-Driven Enterprise

Enterprise AI represents a fundamental shift in both the technology and business landscapes, with an impact as significant as that of software itself.

Consider the impact traditional software has had over the past 30 years. Can you imagine an enterprise of any significance not depending on a huge amount of software? Now consider that Enterprise AI is at the dawn of a new age of software—"Software 2.0" as described by Andrej Karpathy, Director of AI at Tesla.

Software 2.0 refers to the idea that machine learning (i.e., Software 2.0) allows us to extract the rules for solving some problem in our business automatically from data we've collected about the problem. In other words, the modeling process automates the creation of software for us. This can occur in situations in which it would be very difficult for humans to identify, much less program, the complete set of rules that govern the solution to a particular problem.

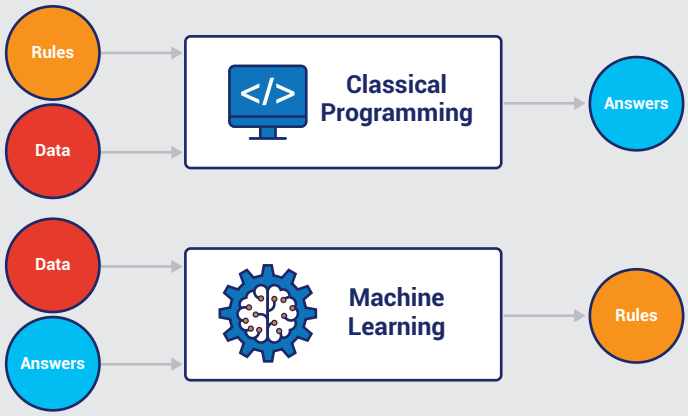

Figure 2. Classical programming vs. machine learning

The widespread adoption of Enterprise AI will have broad implications for businesses; so much so that we believe it will define a new wave of business productivity marked by what we call the Model-Driven Enterprise.

Consider again the role of software in the enterprise, and a few of the major technological and productivity shifts that it has enabled over the past 30 years.

- **The process-driven 90s.** The 90s saw the rise of the process-driven enterprise. During this period of time—which was marked by the widespread application of ideas like Six Sigma, Lean, and business process reengineering—enterprises reorganized themselves around their core processes, collected data about the execution of these processes, and used reports about this data to analyze and improve their performance. ERP systems and business intelligence (BI) tools were among the major enterprise technologies that supported this shift. The connection between data collection and process improvement was a highly manual one, with humans reviewing reports on a relatively infrequent basis (quarterly or monthly reporting cadences were not uncommon), applying fixed rules and intuition, and using the information gained to drive behavior changes.

- **The data-driven 00s.** With Y2K remediation efforts and the introduction of the euro leading to a boom in ERP deployment in the late 90s, widespread digitization of enterprise processes set the stage for a boom in the amount and quality of data collected by enterprises. At the same time, connectivity within and between enterprises became ubiquitous, and the web and mobile came into their own as channels for consumer engagement. Much of the data collected found its way into traditional enterprise data warehouses (EDWs) and eventually into alternatives like Hadoop. While improvements in software development methodologies and the introduction of frameworks like J2EE and .NET, along with the rise of tools like business process management systems (BPMS) and business rules engines, meant that more decisions could be made in software, developing these systems remained slow and risky. Humans remained central to most enterprise decision-making. "Big data" promised to decrease the time lag between insight and action, and was generally successful in getting this reduced from quarters and months down to weeks.

- **The model-driven late 10s and 20s.** In the current model-driven era, enterprises have the opportunity to tap into a fundamentally new source of value creation by putting all the data they've collected to productive use through the creation of machine learning models. These models will be deployed within a wide variety of software applications and systems. This will allow the machines that interact with customers and control back-office functions to make high-fidelity decisions instantaneously. Because these models are created by software through the training process, as opposed to via manual software development, innovation cycles are reduced to days or hours. The technology innovations supporting the shift to model-driven enterprises include deep learning frameworks like TensorFlow and PyTorch as well as a new breed of machine learning platform technologies designed to eliminate friction at various points of the model delivery process.

	Process-Driven	Data-Driven	Model-Driven
Use of Data	Enterprise organized around relatively static processes performed by humans	Incremental improvements; data collected at different points in processes	Data collected at each step of processes used to train next iteration of model
Decision-Making	Fixed rules and intuition used to make decisions	Data used by humans to make more informed, better aligned, and more timely decisions	Data used by machines to make high fidelity decisions instantaneously
Innovation Cycles	Months to years	Weeks to months	Hours to weeks
Timeframe	1990s	2000s	Late 2010s to 20s +

Figure 3. The process-driven, data-driven, and model-driven eras in business

In this light, it becomes clear that Enterprise AI is much more than a fad, but the next logical step in a 50-plus year march of enterprise technology towards supporting improved business decision-making, faster innovation cycles, and improved business performance.

On Digital Transformation

Digital transformation (DX) is a hot topic today among enterprise business and technology leaders. In the context of the changing business/technology landscape, we can see that DX is the process through which enterprises adapt from one era to the next, in the direction of progress. In other words, DX is an ongoing process. As it becomes clear to enterprise leaders that becoming model-driven is in their future, DX will be the banner under which they undertake the journey.

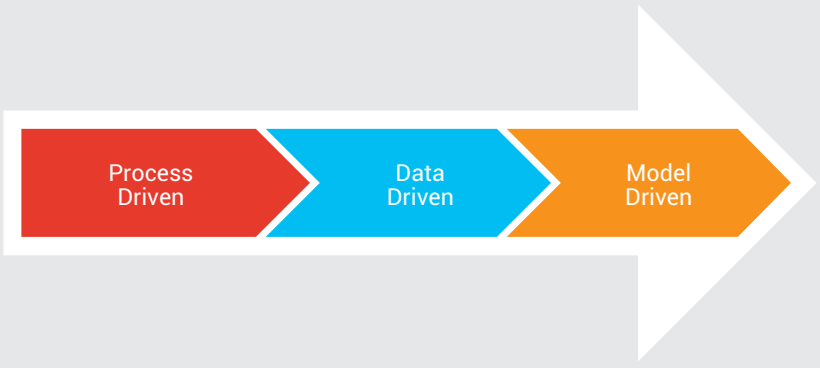

Figure 4. Digital transformation is the ongoing evolutionary process through which the enterprise adopts new technologies

Getting to Model-Driven

As models emerge as a significant source of proprietary business advantage, the ability to create and deliver them to production in an efficient, repeatable, and scalable manner becomes a critical competency.

Key Challenges

As is often the case with other emerging technologies, enterprises face people, process, and technology challenges in their endeavors to efficiently deliver machine learning models to production.

People

Enterprises face a variety of people-related challenges when implementing machine learning and AI. First is the scarcity and cost of experienced data engineers, data scientists, and machine learning engineers. Assuming the hiring hurdle is overcome, numerous organizational and cultural challenges await. Culture is a key factor in the productivity of enterprise machine learning organizations because the organization's approach to problem definition, experimentation, priorities, collaboration, communication, and working with end-users/customers are all guided by organizational culture. The topic of building a strong culture for effective ML/AI is largely beyond the scope of this ebook, although we do return to it briefly in our concluding chapter, Developing Your ML Platforms Strategy.

Process

Developing and deploying models is a complex, iterative process with numerous inherent complexities. Even at small scales, enterprises can find it difficult to get right. The data science and modeling process is also unique (and difficult) in that it requires a careful balance of scientific exploration and engineering precision. Space must be created to support the "science" aspect of data science, but a lack of rigor and automation gets in the way of efficiency. The key is to apply rigor and automation in the right places, and there are many opportunities to do so, as we will see.

Technology

Technology—and its key role in allowing an organization's people to execute its machine learning process more efficiently—is the central focus of this ebook. Technology without process is simply a tool, and while tools can be helpful, their value is incremental. Conversely, process without technology limits the efficiency and automation necessary to scale. It is only by supporting an organization's people—its data scientists and ML engineers in particular—with effective processes and technology, that they are empowered to efficiently apply ML models to extract value from enterprise data at scale.

Driving Modeling Efficiency with ML Platforms

In order to help enterprise machine learning, data science, and AI innovators understand how model-driven enterprises are successfully scaling machine learning, we have conducted numerous interviews on the topic.[3, 4]

The key observation motivating and confirmed by these interviews is that organizations that have successfully scaled ML and AI share a number of characteristics in common. Most notably, they've all invested significantly in building out platform technologies to accelerate the delivery of machine learning models within their organizations. These efforts have resulted in making machine learning more accessible to more teams in the organization, ensuring greater degrees of consistency and repeatability, and addressing the "last mile" of getting models into production and managing them once they are in place.

It's our belief that effective platforms are key to delivering ML and AI at scale. These platforms support data science and ML engineering teams by allowing them to innovate more quickly and consistently.

So, what is a machine learning platform?

A machine learning platform is a set of tools and technologies (backed by a set of practices and processes) established by an organization to support and automate various aspects of the machine learning workflow, including data acquisition, feature and experiment management, and model development, deployment, and monitoring.

Machine learning platforms come in a wide variety of forms. Until recently, they have primarily been found at large technology companies, which have developed their platforms internally, out of necessity, to support increasingly significant investments in machine learning. As the importance of machine learning has become clear to a broader array of enterprises, new commercial and open source ML platform technologies have become available to reduce the barriers to adoption and make the benefits of ML models more accessible.

A fundamental premise of this ebook—and our broader AI Platforms research—is that enterprises should begin their ML platform journey by learning from the efforts of more experienced model-driven enterprises. Thus, to understand the capabilities and benefits available through ML platforms, we next take a look at the platforms established by several early adopters.

ML Platform Case Studies

In this section, we present three representative ML platforms: Airbnb's Bighead, Facebook's FBLearner, and LinkedIn's Pro-ML. Each of these platforms was developed in response to the unique situation, challenges, and considerations faced by its creator. As we will see, each platform took on a unique form as a result of these influences, while at the same time many common patterns are evident.

Airbnb's Bighead

Machine learning has long been used to power Airbnb's core accommodations marketplace with applications like search ranking, smart pricing, and fraud prevention. In 2016, only a few major ML models were in production and it typically took teams eight to twelve weeks to build new models. The Airbnb machine learning technology stack at the time was based on Spark, Scala, and a homegrown machine learning library called Aerosolve.

In 2016, the company's engineering team realized that in order to meet its business goals, it would need to dramatically increase the velocity with which it was putting machine learning models into production. There was a growing desire on the part of Airbnb's development teams to take advantage of rapidly evolving third party ML tools like TensorFlow, PyTorch, and SciKit Learn. They also experienced frustrating discrepancies between the results they were seeing in offline training and online serving, preventing them from meeting desired performance objectives and eroding confidence in their machine learning models.

To address these issues, Airbnb established an ML infrastructure team in 2017. The team describes its mission as eliminating the *incidental* complexity of machine learning—getting access to data, setting up servers, and scaling model training and inference, allowing developers and data scientists to focus on dealing with its *intrinsic* complexity—identifying the right model, selecting the right features, and tuning model performance. In establishing this team, they empowered more users to build ML products, reduced the time and effort required to do so, and produced more consistent results with the models they put into production.

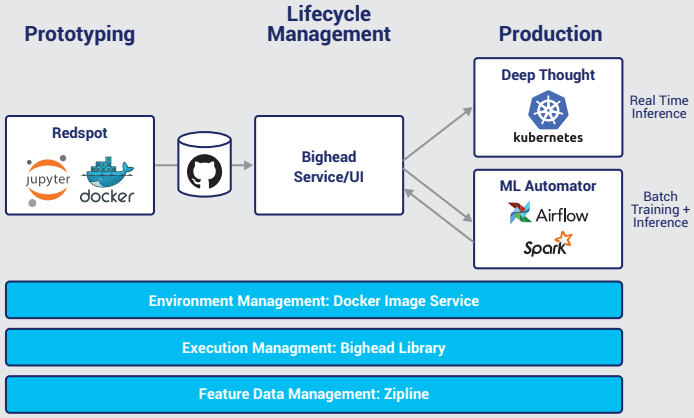

Figure 5. Airbnb's Bighead

Airbnb's internal ML platform is called Bighead. Bighead is an end-to-end platform for building and deploying ML models that aims to make the machine learning process at Airbnb seamless, versatile, consistent, and scalable. It is built in Python and relies on open source technology like Docker, Jupyter, Spark, Kubernetes, and more. These open source components are customized and integrated for Airbnb's specific needs. Like much of Airbnb's technology infrastructure, Bighead runs in AWS.

At the time of our interview, the platform was supported by an ML infrastructure team of 11 engineers and one product manager. In the fall of 2018, Airbnb announced its plans to open source parts of Bighead and Zipline in early 2019, but this hasn't yet materialized.

Bighead consists of the following major components:

Zipline

Zipline is a feature data management abstraction that sits in front of Airbnb's data warehouse. Apache Spark powers many of its features, especially those performing offline tasks such as training set backfills and feature computation. Zipline allows Airbnb's ML to more easily and consistently access feature data for ML models. It provides:

- A feature repository providing vetted crowdsourced features at different levels of granularity such as host, guest, listing, or marketplace[5]
- An easy-to-use configuration language for defining new features
- Efficient point-in-time data backfills
- Feature exploration and visualizations
- Automatic data quality monitoring
- Consistency between feature availability for offline training and online scoring (batch and streaming)
- Explicit ownership of feature data pipelines

Airbnb has reported that Zipline has reduced the time that its ML practitioners spend collecting data and writing transformations for machine learning tasks from months to days.[6]

Bighead Library

Bighead Library is the core execution management and data transformation library in Bighead. It allows users to define machine learning workflows for data preprocessing, training, inference, model evaluation, and visualization in a standard graph-based (i.e., directed acyclic graph, or DAG) format. Bighead Library pipelines are reusable, composable, and shareable.

A variety of popular machine learning frameworks are supported in Bighead Library pipelines including TensorFlow, PyTorch, Keras, MXNet, Scikit-learn, XGBoost, H2O, R, and more.

In addition, Bighead Library provides a catalog of over 100 different canned transformations that can be applied to data in a variety of standard formats such as text or images, encouraging users not to "reinvent the wheel."

Pipelines built with Bighead Library propagate feature metadata end-to-end so that users can understand feature provenance and visualize feature importance at the end of their pipelines.

Redspot

Redspot is a centrally managed, multi-tenant Jupyter Notebook service available to Airbnb's machine learning practitioners. It is a fork of the official JupyterHub project that is tightly integrated with the rest of the Bighead ecosystem.

Redspot lets Airbnb's ML practitioners use notebooks as first-class tools by providing easy access to data and pipelines, and allowing them to deploy notebooks all the way to production.

Every user's environment in Redspot is containerized via Docker containers. This allows them to install systems or Python packages without affecting other users. Users can use vetted base images from Airbnb's in-house image repository or create new images using provided build tools.

Redspot notebooks are version controlled, can be reverted to prior checkpoints, and can be easily spawned on dedicated AWS CPU and GPU instances. Notebooks and files are easily shared via AWS EFS.

Bighead Service

Bighead Service establishes the core modeling framework for Bighead and provides centralized model lifecycle management and experiment management services. It provides a single source of truth and history for models at Airbnb, whether they're in development on a developer laptop, or running in a production cluster.

Bighead Service keeps track of "Model Versions" consisting of the model's code and a Docker image through which it can be run, and "Model Artifacts" consisting of a set of model parameters such as those learned via training. Instantiated Bighead models can be accessed via a lightweight standardized API baked into the various Docker images.

Bighead Service also provides a user interface for:

- **Experimentation.** Users can set up and track online model experiments.
- **Evaluation.** Users can access a dashboard of model metrics, visualizations, and alerts.
- **Deployment.** Users can track model changes, and deploy and roll back models.

Deep Thought

Deep Thought is a scalable serving environment for Bighead models. It is based on containers and Kubernetes, providing consistent development and training environments, runtime isolation, and scalability. It is completely configuration-driven so data scientists don't need to involve engineers to deploy new models. Deep Thought exposes deployed models via APIs, and provides standardized logging and alerting accessed through Bighead Service dashboards. Models have access to feature data from Zipline.

ML Automator

ML Automator is a workflow engine that runs behind the scenes to automate common offline tasks in Bighead such as periodic model (re)training and evaluation, batch scoring, uploading scores, and creating dashboards and alerts based on scores. ML Automator users specify these tasks declaratively and ML Automator generates Airflow DAGs under the covers, specifying the appropriate connections to Bighead resources and Zipline data. Computation is run on Spark for scalability.

Facebook's FBLearner

Machine learning is at the heart of many Facebook services including the newsfeed, ads, search, translation, speech recognition, and content understanding. The ubiquity of machine learning at the company is due in large part to a 2014 initiative that sought to re-envision its machine learning development process from scratch with the goal of putting state-of-the-art ML and AI algorithms in the hands of every engineer.

By mid-2016, the organization unveiled FBLearner Flow,[7] what we now understand to be the first (and central) element of its broader ML-as-a-Service platform. FBLearner was designed with the guiding principles of reusability, scalability, and ease-of-use. At the time of its announcement, FBLearner had already had a profound impact at Facebook with approximately 150 authors creating and publishing machine learning workflows. These workflows were used by over a quarter of the engineering team and generated over 6 million predictions per second. Today, most machine learning pipelines at Facebook are run via FBLearner. At the time of our interview, FBLearner and Facebook's other AI Infrastructure initiatives were supported by a team of 17 people.

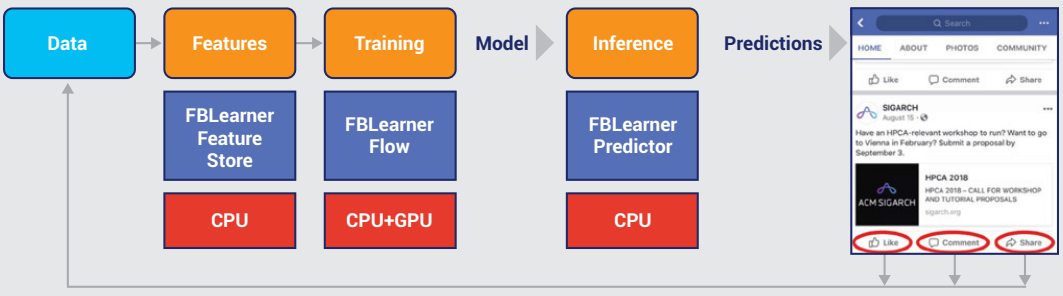

Figure 6. Facebook's FBLearner[8]

FBLearner consists of these key components:

FBLearner Feature Store

FBLearner Feature Store is the component of the FBLearner platform that allows developers to discover and use pre-built features in their models.

Feature Store is essentially a catalog, or marketplace, of feature generators for use in both training and serving scenarios. Teams at Facebook can share, discover, and consume these features to accelerate the development of their models.

The feature generators offered in Feature Store expose data in the Facebook data warehouse in a consistent manner. Feature Store provides a rich type-system for describing this data, ultimately enabling many of the downstream features of FBLearner Flow.

FBLearner Flow

FBLearner Flow is Facebook's platform for building, training, and evaluating machine learning models. It is organized around three central concepts:

- **Workflows.** A workflow is a single pipeline defined within the FBLearner Flow system. Workflows describe the steps necessary to train or evaluate a specific model in FBLearner. The platform itself is agnostic to the type of ML algorithms used. Any engineer can write a workflow for their favorite algorithm and publish it to the entire team.

- **Operators.** Operators are the basic components from which workflows are composed. They are the smallest unit of execution in FBLearner. They have distinct inputs and outputs and run on a single machine. Operators are designed to be a very flexible abstraction. Generally, they are functions in a Python program, but essentially anything for which a binary can be provided can be turned into an operator.

- **Channels.** Channels represent the flow of data between operators within a workflow. They are characterized by their inputs and outputs and must have a well-defined type.

These concepts are implemented via three core components:

- **A workflow authoring and execution environment.** Workflows are defined in Python code with special, platform-specific extensions. Using these extensions, developers define graphs of operators and specify these operator's parameters such as inputs, outputs, models, and evaluation metrics. Each operator declares its CPU, GPU, and memory requirements. At runtime, the number and type of input data are validated, workflows are parallelized for scalability, and they are scheduled to run on a slice of a machine that meets the requirements.

- **An experimentation management dashboard.** FBLearner Flow provides a user interface for launching and managing workflows and experiments, viewing and comparing experiment results, and deploying models to production. The metadata and results of

all workflow runs are stored and indexed, allowing engineers to easily find experiments via any number of parameters. The system supports complex hyperparameter sweeps for model tuning and provides visualizations to help developers determine the configurations that produced the best results.

- **Predefined pipelines.** Flow provides a library of user-contributed machine learning workflows as well as a set of scalable workflows maintained by the company's AI Infrastructure team. The workflows can be applied to commonly used models like deep neural networks, gradient-boosted decision trees, support vector machines, and logistic regression.

FBLearner Predictor

After training, finalized models from FBLearner Flow are then deployed to production via FBLearner Predictor. Predictor provides a scalable, low-latency, multi-tenant model serving environment for online predictions based on live traffic. Using the integration between the Flow UI and Predictor's model serving capability, users can run experiments in which multiple versions of live production models are deployed and compared.

Related work

Facebook's ML platform engineering team has developed offerings for each layer of its machine learning stack. At the hardware level beneath FBLearner, they have created custom server designs (which they have open sourced via the Open Computing Project). At the framework level above FBLearner, they lead the development of PyTorch and are a leading contributor to the ONNX ecosystem for framework interoperability.

At the FBLearner level, Facebook has published only high-level descriptions of the system and its capabilities. Because of Facebook's immense scale and unique internal systems it is unclear how much of FBLearner would generalize to third parties. Despite this, the broader ideas behind it continue to influence other systems and have certainly informed the other platforms and tools described in this ebook.

LinkedIn's Pro-ML

Machine learning is in wide use at LinkedIn, powering everything from user-facing personalized news feeds and job recommendations to back-office functions such as advertising and sales lead generation.[9, 10, 11] In order to empower LinkedIn's disparate teams to use machine learning more efficiently and productively, the company established a platform engineering team whose primary product today is an internal machine learning platform: LinkedIn Pro-ML.[12]

Pro-ML is the latest step in LinkedIn's ML platform journey. As their platform team has developed and expanded, they have successively standardized and outgrown a number of platform technologies. Initially, LinkedIn began with a Hadoop-based platform, then moved to Apache Spark and MLlib. Later, they developed and open sourced their own MLlib replacement: Photon-ML.

In parallel with LinkedIn's evolving technology stack, the platform team's audience has also changed. The team initially catered to the company's core set machine learning developers. However, as in-house efforts such as its AI Academy broadened the base of developers working with AI and ML, LinkedIn's platform team stepped up to support the needs of this broader audience. Pro-ML is a suite of complementary systems that help streamline the model development lifecycle at LinkedIn. Pro-ML consists of:

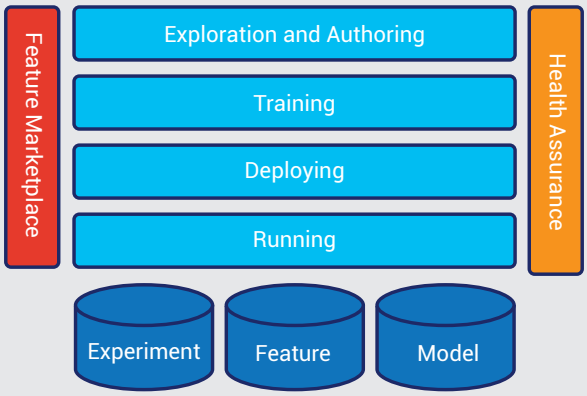

Figure 7. LinkedIn's Pro-ML

Feature Marketplace

LinkedIn observed that many of their teams were developing similar features, creating a "pipeline jungle" that was too complex to build upon. The Feature Marketplace was developed to address this challenge, providing a way for teams to simplify and unify the features available on Pro-ML. Feature *providers* can publish their features using simple configuration files that allow them to specify the location of their source data, such as an HDFS path or a pointer to a key-value store, along with the requisite extraction logic. The logic that provides the feature is hidden behind an abstraction layer, allowing *consumers* to utilize features simply by name.

Exploration and Authoring

Pro-ML provides a standardized suite of tools for model development (such as Jupyter Notebook) along with a pre-built set of ML models that developers can use off-the-shelf. Developers can specify how their models are to be trained on features from the Marketplace, as well as on custom features, using a DAG specified in code.

Training

Models in Pro-ML can be built using Tensorflow or LinkedIn's own Photon ML. Photon ML is an open source ML library that their algorithms team developed for Spark. It offers many models in popular use at LinkedIn such as Generalized Linear Models and Generalized Linear Mixed Models.

Hadoop and Apache Spark provide the foundation for training models at LinkedIn with the former supplying distributed storage (via HDFS), and the latter providing the runtime platform for large-scale training and inference. LinkedIn developed and open sourced TonY (Tensorflow on YARN) which helps effectively manage clusters running distributed TensorFlow.

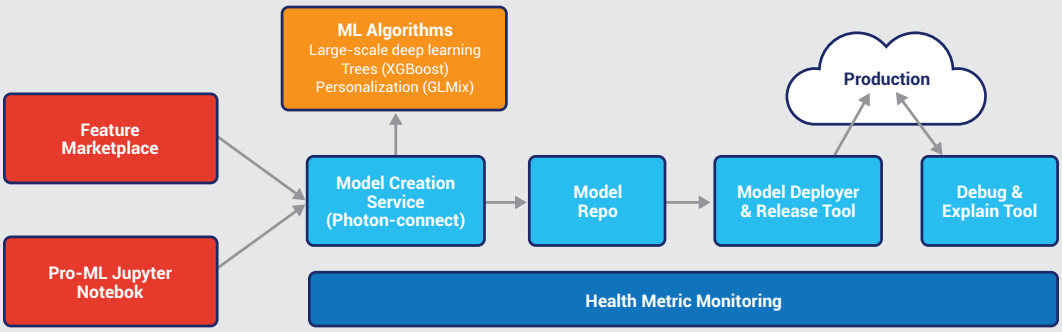

Figure 8. A workflow-oriented view of LinkedIn's Pro-ML

Deploying

LinkedIn's production models can be very large and often need to be distributed over multiple machines to perform adequately. LinkedIn has developed workflows and tooling that ensure new models can be deployed and released into production in a consistent and controllable way. This provides developers with a standardized way of deploying and monitoring their models while ensuring a high quality of service for production systems.

A web interface called Model Explorer allows users to discover, publish, and deploy models into production. It also provides the ability to perform model validation in a test environment. Once validated, models can be deployed to a single instance. Once validated on a single instance, the model can then be scaled to many production instances.

Health Assurance

Pro-ML continuously monitors feature distributions to detect changes. Monitoring includes mean, variance, and some higher-order statistics. If the feature distribution exhibits unexpected behavior, the system automatically generates alerts. When an alert is received, the explanation of why the model misbehaved can be investigated.

Understanding ML Platform Capabilities

Presented with case studies drawn from the experiences of tech giants and ML/AI early adopters, it's easy for enterprises that don't fall into either of these categories to ask: what's this got to do with me? After all, the typical enterprise doesn't have near the scale of a Google or Facebook when it comes to ML modeling.

Yet, while factors like team size, business models, technological maturity, available resources, and regulatory considerations vary among organizations and industries, there is much that can be learned from the platforms built by these early model-driven enterprises.

Each of the platform features or components they've built points to a lesson they've learned about overcoming the obstacles to putting machine learning models into production and enabling greater throughput and effectiveness for their data scientists and ML engineers.

By abstracting across the experiences of many early platform builders and users, and identifying the capabilities that we see frequently recurring in the platforms that they've built, we have identified a core set of platform capabilities. These capabilities, or requirements, for scaling machine learning in the enterprise are presented and organized here according to our high-level model of the machine learning process. Notes associated with these capabilities provide additional context and examples from the open source and vendor ecosystems.

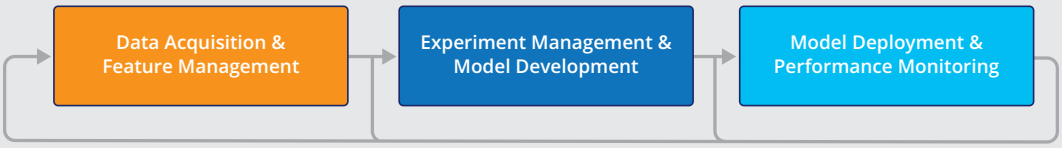

Figure 9. The machine learning process

While the typical company might not need the full set of capabilities that a Facebook or Airbnb has implemented, there is certainly much we can learn about operationalizing and scaling machine learning from their examples.

Data Acquisition & Feature Management

Enterprise machine learning is made possible by the large and growing amount of data now captured by the typical company. But simply *having* the data is not enough. Successfully scaling machine learning in the enterprise depends on an organization's ability to effectively harness its data resources in an efficient and repeatable manner. The organizations that do this most successfully have processes and technology in place supporting:

Centralized data access. Organizations with modern data warehouses, data lakes, or data fabrics are at a significant advantage when it comes to scaling their ability to deliver ML projects. Without centralized data or data access, finding and gaining access to the data needed to build machine learning models can consume a great deal of time and effort. On the other hand, when data is centralized, teams no longer need to navigate or search multiple systems to access the data they need for their projects. As a result, any efforts resulting in simplified data access tend to be a force-multiplier on data science resources. Furthermore, a data warehouse can also be a useful place to store transformed data and features, facilitating reuse across projects and teams.

The line between the modern data warehouse and the data lake continues to blur. It is common to see systems based on the Hadoop ecosystem here, such as Apache Hive or Impala. For organizations all-in on cloud, common choices include a combination of cloud storage plus Amazon Redshift, Google BigQuery, or Microsoft Azure Cosmos DB, or Snowflake. For centralized access to distributed data, Alluxio is one of several available commercial solutions.

Repeatable data pipelines. Enterprise data rarely exists in the exact form or format required by data scientists for a given project. Rather, raw data must be processed through a series of transformations in order to cleanse and normalize it before it can be used for training. Once a model is put into production, this same sequence of transformations must be applied to the data to ready it for inference.

Early in the exploratory phase of model development, these transformations are often applied in an ad hoc manner. However, manual transformations should give way to programmatically executed transformations very quickly, as the former are highly error-prone, not readily repeatable, and don't scale.

Automation via scripts or other programs can be a starting place, but more sophisticated scenarios are better supported by DAGs, which are in use supporting data pipelines at each of the organizations we've profiled. Apache Airflow is an open source workflow framework that supports DAGs and can be used to build data pipelines. Apache Spark is also frequently used for data pipelines. Pachyderm and DVC are startups with more full-featured open source offerings in this area.

Data labeling. Most ML in use today is supervised learning, meaning it requires labeled training data in order to work. Sometimes these labels can be extracted from existing data, but often the labels we need to support ML applications must be manually created. Labeling is often thought of as something that happens separately and before the machine learning process itself, but this is changing for several reasons. First, the collection of labeled data is very expensive, and integrating it into the ML process ensures that only the amount and type of data needed to meet the project's goal is labeled. Perhaps more interesting is the increasing maturity of techniques like active learning and semi-supervised learning. These techniques place labeling squarely in the ML loop and use ML itself to determine which data to label. In any case, efficient labeling requires tooling customized to the data and labels for a specific problem.

Vendors offering software tools to facilitate labeling and annotation include Alegion, Figure Eight, Mighty AI, and Scale. These companies' offerings include both annotation tools as well as the ability to manage labeling projects and external human labelers. Explosion AI's Prodigy and Alectio (still in stealth at the time of this writing), offer tools based on active learning for increasing the data-efficiency of models.

Feature repository. A given organization's models often share the same or similar features. For example, many machine learning projects at a retailer will use features representing customers, products, orders, etc. Constructing a library of common features allows the organization's data scientists and developers to avoid reinventing the wheel each time they want to incorporate these basic business entities into their models. Such a library or repository is often called a feature store, reflecting its role as both a storage facility for features, and a marketplace of sorts where users can share, discover, and consume new features.

Back in 2017, Uber reported having over 10,000 features in the Michelangelo (its ML platform) feature store.[13] Today, the feature store remains somewhat immature in practice, with few commercial or open source offerings available. As noted earlier, Airbnb announced its intent to open source Zipline, but hasn't yet delivered on this promise. Google and GO-JEK (the $10 billion Indonesian delivery and transportation startup) announced Feast,[14] an open source feature store. Hopsworks[15] by Logical Clocks is an open source, Hadoop-based end-to-end ML platform that includes feature store functionality.

Feature provenance. Unlike the capabilities mentioned so far in this section, feature provenance isn't so much a system. Rather, it's the *idea* that with the right technology and processes in place—including centralized data, repeatable data pipelines, and a feature repository—it is possible to trace all the way back from a given inference or model to the feature data incorporated into the inference decision or the model's training.

Pachyderm touts provenance as a key feature. Provenance also figures into a general set of model governance features in the ParallelM, an end-to-end ML platform recently acquired by DataRobot. Provenance[16] also happens to be the name of a Python library that can track how and where artifacts such as models, features, or any object or file are used in an ML pipeline.

Backfills. Model development frequently results in new features being identified and created. These new features can involve complex and computationally expensive data transformations. In order to move these models into production and support fast inference, we often want to precompute, or backfill, the values of these features for our historical data, and store them in the data warehouse. The ability to programmatically perform backfills can help ensure that this is done in a repeatable and efficient manner. To date, we've not come across any dedicated tools focused on automating backfills. Workflow tools such as Apache Airflow can provide a useful framework for implementing your own programmatic backfills.[17]

Data versioning. Achieving the highest levels of reproducibility and the ability to forensically analyze deployed models and their decisions requires being able to go back in time to determine the exact state of the training data at the time a model was trained.

Pachyderm and DVC, both mentioned previously, are specialized tools with robust data versioning capabilities. Several end-to-end ML platform offerings, notably Dotscience (still in stealth at the time of writing) and MissingLink also include data versioning capabilities.

Experiment Management & Model Development

Data science is a scientific endeavor and requires experimentation in order to identify and optimize a predictive model. Managing these experiments via a software tool or system allows data scientists to get experiment parameters and results out of spreadsheets (or worse, file names, post-it notes, or their heads) and into a tracking system aiding repeatability, collaboration, optimization, and automation. The organizations we've profiled have all built robust experiment management and model development features into their platforms to allow data scientists to delegate repetitive aspects of model training and optimization to machines, freeing their time to focus on high-value creative activities. Here are some of the most common capabilities of these systems:

Experiment tracking and visualization. Machine learning model development is an inherently iterative process. The process of developing models typically involves running a series of experiments—hundreds or thousands are common—in order to identify the model types and parameters that result in optimal performance. Historically, the parameters that define each experiment—that experiment's hyperparameters—are manually tracked by data scientists. This is done via any one of a variety of methods including using a lab notebook, document, spreadsheet, file and folder naming conventions, log files, etc. The experiment management features of modern ML platforms eliminate the burden and fallibility of manual tracking by programmatically logging the parameters and results of each test run.

With experiment parameters and results readily accessible, access to visual plots can be provided to allow data scientists to easily compare the performance of different model versions. These plots can be generated automatically as each experiment is logged and made available to data scientists via an experiment results dashboard.

Some degree of support for experiment management and tracking is included in most end-to-end ML platform offerings. In addition, a number of specialists exist as well. Comet, Neptune, and Weights & Biases all offer full-featured experiment management and visualization solutions via SaaS, with the latter specializing in deep learning. SigOpt[18] pairs experiment tracking and visualization with enterprise-grade automated hyperparameter optimization (more below). Sacred[19] is an open source tool for configuring, organizing, logging and reproducing experiments developed at IDSIA. TensorBoard,[20] part of the TensorFlow ecosystem, has emerged as a popular open source tool for logging and visualizing experimental results. It can be used independently of the TensorFlow framework with a compatible logging library.

Model version control. In addition to tracking model parameters, trained models themselves can also be tracked across the model development lifecycle via a version control system. Versioning models allows data science and machine learning teams to more readily reproduce experiments and provides for greater consistency between prototyping and production environments.

Models may be versioned by checking model code, serialized model objects (e.g. a Python pkl file), a model's Docker container, or other model artifacts into a Git repository, ideally automatically at appropriate points in the modeling workflow. Most modern ML platform offerings support versioning models. Paperspace and Valohai are examples of startups whose platform products emphasize model versioning. Verta is a specialist in this area; the company's founder created the open source ModelDB[21] model management system while in graduate school.

Distributed training. For smaller ML models, training can be reasonably done on the data scientist's desktop. As organizations begin working with more sophisticated models such as deep learning, training against larger and larger datasets, or enforcing more sophisticated data access controls, distributed training in a centralized environment becomes increasingly important. Distributed training frees up the data scientist desktop and provides a centralized nexus for management and control.

Many homegrown, commercial, and open source ML platforms rely on the open source Kubernetes container orchestration system for distributed training. For more details on how Kubernetes can be used to support data science—including a look at some representative tools such as the Kubeflow[22] ML platform subproject—see the first ebook in our AI Platforms series, *Kubernetes for Machine Learning and Deep Learning*.[23]

Automated feature generation. Feature engineering, the process of creating new features to train machine learning models, is historically the most challenging and time-consuming aspect of data science. At its heart, feature engineering involves iteratively applying a series of transformations and aggregations to existing data. Examples include generating derived features (such as calculating age from a birthdate) or converting categorical variables (such as transaction types) into one-hot encoded, or binary, vectors. Some of these operations are based on a data scientist's experience, domain knowledge, and intuition. Others are really just the rote application of patterns that are common to a specific type of problem or data, and can thus be automated.

One technique for automated feature engineering, called Deep Feature Synthesis,[24] was created at MIT's Computer Science and Artificial Intelligence Lab (CSAIL) and is being commercialized by startup Feature Labs via the open source, Python-based Featuretools[25] project. Automated feature generation is also commonly included as part of AutoML tools, to be discussed shortly.

Automated hyperparameter optimization. Complex models depend on many tunable parameters that must be optimized to maximize performance and accuracy. Rather than manually experiment with various combinations of values, automated hyperparameter optimization (HPO) tools systematically work to identify the optimal hyperparameters. HPO can use simple inefficient methods such as random or grid-based search of the hyperparameter space to look for optimum parameter values, or rely on more sophisticated techniques—like Bayesian optimization—to optimize parameters more efficiently.

Many end-to-end platforms include some kind of HPO capability, often based on one of the many[26] open source hyperparameter tuning packages like Yelp's MOE,[27] SMAC,[28] Skopt,[29] Katib,[30] BayesOpt,[31] Hyperopt,[32] or Tune.[33] SigOpt, built by the creators of MOE, SMAC and BayesOpt, provides a full-featured commercial solution for black-box hyperparameter optimization offering advanced features such as mixed parameter spaces, high hyperparameter dimensionality, asynchronous parallelization, multiple metrics,th and an API to enable the service.

AutoML. AutoML is a superset of automated feature generation and HPO that employs a variety of techniques to identify useful features and the best models and model architectures. In the case of relatively simple ML problems (e.g. classification and regression on tabular data), AutoML tools iteratively generate features and apply and optimize appropriate models and compare them against one another until a winner is found. For more involved problems such as deep learning, complex techniques such as neural architecture search and neuroevolution may be employed.

The term AutoML was arguably popularized by **Google** with a late 2017 *New York Times* PR coup[34] followed by the release of Google Cloud AutoML Vision in early 2018. Google has since expanded its line of Cloud AutoML services[35] to include AutoML Video Intelligence, AutoML Natural Language, AutoML Translation, and AutoML Tables. Software vendors **DataRobot** and **H2O** have mature software offerings targeting traditional machine learning use cases. For deep learning, the **Determined AI** end-to-end offering includes an AutoML feature based on neural architecture search.

Workflows. Throughout the training process, a number of experiments are run, artifacts created, and models evaluated. At scale, this creates a lot of busy work and opportunity for error, as data must be moved around and formatted appropriately for each step. Standardized, automated workflows are an important element of ensuring the efficiency and repeatability of training.

The Apache Airflow project provides a strong foundation upon which to build and execute ML workflows. For Kubernetes users, Kubeflow provides a container-based solution with its Pipelines[36] feature. End-to-end ML platform offerings typically have a strong notion of workflow. This can be an advantage for users just getting started but can pose problems for those with existing processes they'd like to automate. See the discussion on Wide vs. Deep platforms in the section, "Developing Your ML Platforms Strategy."

Coding environment and standards. Clear standards around the type of models and frameworks supported by the platform and how models are to be coded are expedient. Jupyter Notebook is ubiquitous among those with a data science orientation, but those with more of an engineering background may prefer standard code modules and development tools.

The open source Papermill[37] project, developed and used at Netflix, allows users to parameterize and execute Jupyter notebooks. The team developing the popular Fast.ai deep learning library

used a similar technique in the development of its latest version: all code is developed in Jupyter notebooks and extracted upon code check-in. Startup Fiddler was founded to build a platform for machine learning that incorporates traditional IDE concepts such as debugging. Dataiku DSS allows models to be prototyped using a visual drag-and-drop interface.

Whatever the programming environment of choice, it is common to provide extensions to that environment to simplify tasks like data access and expose various features of the machine learning platform.

Framework support. Different organizations take different stands on the issue of how opinionated the ML platform should be. Some organizations believe that developers should be free to make whatever choices are best for their applications and that the platform should support all popular languages and frameworks. Others choose to provide deep support for a single framework (open source or internally developed), to the exclusion of others. Both choices are valid and depend on the goals and resources of the organization.

Airbnb, for example, is an organization whose platform team seeks to be framework agnostic.[38, 39] Twitter, on the other hand, has recently standardized on TensorFlow for the most recent version of its DeepBird deep learning platform.[40]

Model Deployment and Performance Monitoring

The ultimate goal of our data science efforts is the creation of a model, right? Not exactly. The goal is a model *put to productive use*. This requires overcoming the "last mile" barriers to getting the model into production.

While most of the data science process is performed outside of the critical path of key business systems, models deployed as part of production systems are the opposite. They must be able to generate their predictions within time, reliability, and (in the case of edge deployment) power bounds appropriate for the systems in which they're being deployed.

Because of the potential for differences between training data distributions and those of real-world users/traffic, attention must also be paid to the way we introduce production traffic to new models. Further, models have a limited shelf life and must be continuously monitored for degradation.

These needs and more can be met by the model deployment and management capabilities of ML platforms:

Model serving. A basic requirement of an ML platform is being able to serve models in a scalable and consistent manner. The organizations we've interviewed have typically addressed the need to support inference for multiple models with a single, multi-tenant environment to which any of the organization's models can be deployed. Different organizations and platforms standardize on different types of model artifacts for deployment, including binaries, code, parameters, and full containers. Deployed models are often exposed as gRPC, REST, or messaging-based decision microservices via API wrappers.

Kubernetes is a popular choice for distributed inference environments due to its built-in container and microservice management features like routing, load balancing, and declarative autoscaling.[23] Algorithmia's AI Layer allows organizations to publish decision services using Git and Docker, deploy them to any cloud or on-premises infrastructure using Kubernetes, and manage their use down to the API call level. Seldon Core is an open source model serving framework for Kubernetes that is incorporated into platforms like Kubeflow, IBM Fabric for Deep Learning[41] (FfDL), and Red Hat OpenShift. Cloud-based ML platform offerings like AWS SageMaker, Microsoft Azure Machine Learning, and Google AI Platform typically support one-click deployment of trained models.

Model instrumentation and evaluation. As previously noted, models are perishable. The statistical distribution of data that a model sees in production and that of the training data will tend to drift over time. This will cause model performance to degrade and, if this drift isn't detected, can result in negative business outcomes such as lost sales or missed fraud. To prevent this, the model serving apparatus should provide a variety of ongoing monitoring capabilities. Ideally, basic monitoring should be applied automatically and transparently by the deployment platform, and engineers should be given tools or APIs that allow them to extend this to meet model- or application-specific requirements. Basic monitoring should include, at minimum, logging all scoring features and decisions for later analysis. Dashboards should collect and display model performance over time in both technical and business terms to simplify problem identification by both business and technical users.

Most platform offerings with support for deploying models also offer some type of visualization. Dataiku DSS, for example, provides dashboards and data validation policies for in-production models, allowing users to monitor model performance metrics, drift, data consistency, and more.

Model data validation. More sophisticated platforms continuously monitor the statistical properties of data seen by production models and compares these to the model's reasonable operating range, as determined by the training data distribution. When drift exceeds a predetermined model tolerance, the system can take remediating action such as issuing alerts or flagging the model for retraining. Beyond statistical drift, many errors seen in production stem from a disconnect between the features expected by the deployed model, and those delivered by the production data pipeline. This is often the result of these subsystems being developed independently. The model deployment platform can play a role here by monitoring the presence, type, and statistics of features presented to the deployed model to assure that the model's expectations are met.

There is an excellent discussion of data validation issues with production models in *Data Management Challenges in Production Machine Learning*[42] by authors at Google.

Phased deployments and online experiment management. Platforms can help ensure safe and effective model deployment by automating the implementation of phased deployments and the management of online experiments. These two share direct analogs with the deployment strategies DevOps teams have been refining to support the safe rollout of microservices. Blue-green testing allows for the quick rollback of new models that fail on deployment. Canary deployments allow for testing of new models with small amounts of traffic until they're validated. And A/B testing allows for competing models to be tested on live traffic. All of these techniques rely on version-aware automated model deployment mechanisms, as well as load balancing and traffic management features provided by the platform.

For example, TensorFlow Serving, the model serving component of the TensorFlow Extended (TFX) platform, uses abstractions like sources, loaders, version policies, labels, and aliases to flexibly support arbitrary A/B testing and canary deployment schemes.[43, 44] AWS SageMaker supports similar functionality using "endpoints."[45]

Batch scoring. Some applications require that developed models be used to score large amounts of data offline rather than scoring online via a microservice. If offline scoring needs to be done with regularity it may be integrated via the platform to help ensure that consistency requirements such as model version and data validation are met.

Airbnb's ML Automator, for example, supports batch inference using a layer built on top of Apache Airflow and Spark.

Summary

In this section, we've reviewed several capabilities commonly provided by machine learning platforms. Next, we explore how to get started assembling your organization's own ML platform.

Data Management	Experiment Management	Model Management
• Data lake/data warehouse • Automated/declarative transformation pipelines • Feature store/marketplace • Feature data snapshotting/ time-machine/backfills • Feature provenance	• Notebooks • Flexible support for arbitrary algorithms/ frameworks • Model pipelines/workflows • Offline experiment management • Distributed training • Hyperparameter optimization	• Model data validation • Model instrumentation and evaluation • Model serving • Online experiment management
Multitenant training & serving infrastructure: • Docker/Containers/Kubernetes • Commodity compute and/or cloud • See our ebook, *Kubernetes for Machine Learning and Deep Learning*, for more on this		

Figure 10. Summary of ML platform capabilities

Developing Your ML Platforms Strategy

So far we've discussed the importance of developing a model-driven orientation for your enterprise, and how that creates the need to be able to deliver models to production rapidly and consistently. We then explored some examples of the internal platforms that leading model-driven companies have developed to help them achieve this goal. From these, we identified a set of common capabilities that define the modern machine learning platform.

The next question becomes: where do we go from here? How can you use this research to develop an actionable plan for building out your own enterprise machine learning platform? Or, more ambitiously, for helping your enterprise achieve model-driven excellence?

Here, we present seven steps or considerations that will help you develop your organization's ML platform strategy. By documenting the application of these considerations to your enterprise, you will be well down the road towards articulating an ML platform strategy for your organization.

1. Know your why

Organizations that have deployed ML platforms cite a wide variety of benefits that help justify their investment. Conversely, by starting from the benefits that are most important to your organization, you will quickly get a better idea about which platform capabilities and characteristics will be most important for you. Here are a few of the benefits that come up most often:

- **Abstraction.** In small data science organizations, the same data scientists are responsible for all aspects of machine learning. As the organization matures, more specialized roles evolve and it becomes important to provide for a separation of concerns. At scale, for example, we don't want data scientists or ML engineers dealing with setting up the infrastructure upon which their models are trained and run. These tasks are best handled by ML infrastructure specialists. Platforms provide a leverage point for enforcing this separation of concerns.

- **Agility.** Agility speaks to the speed with which an organization is able to innovate and adapt to changing needs or new trends. In a fast-changing, model-driven world, there are always more models needed than the organization has the capacity to produce. Furthermore, those models that have been developed tend to degrade over time. By accelerating their ability to get new models into production, ML platforms help enterprises become and remain more competitive.

- **Automation.** Many aspects of the enterprise machine learning process are well-defined, repetitive, and exacting. These are ideal candidates for automation, which helps ensure greater throughput and consistency. With repetitive tasks reliably delegated to the ML platform, data scientists and developers can focus their attention and intellect on the more valuable challenges of problem and solution definition.

- **Consistency or repeatability.** Consistency and repeatability throughout training and production are key requirements for any mission-critical ML use cases and are key benefits offered by ML platforms. ML platforms can help ensure that the correct data is available to models both in training and in production, that the model in production is indeed the model the organization thinks is in production, and that the data seen in production is similar enough to the data that the model was trained against.

- **Democratization.** As organizations embrace the goals of becoming more model-driven, it can be expedient to widen the circle of individuals empowered to build machine learning models. ML platforms help democratize machine learning by hiding much of the incidental complexity of model training and deployment.

- **Governability.** In enterprise environments, governance requirements such as data security, privacy, reproducibility, explainability, fairness, and compliance will regularly come into play. Leaving it to each team or project to determine how to solve these problems is inefficient, wasteful, and goes against many of the fundamental tenets of governance. Your ML platform can provide a structure within which these teams can operate, helping to ensure greater governability.

- **Performance.** Models that perform well on a given task are the goal of a data scientist's experimentation and the desired outcome of the modeling process as a whole. Performance is often the difference between a model that makes it into production and one that does not. By allowing teams to more easily find and access the data and features needed for models, automating model selection and hyperparameter tuning, and monitoring in-production models for performance degradation, ML platforms can help teams build and maintain high-performing models.

- **Productivity.** Because of the highly iterative nature of machine learning, each step in the process that can be accelerated or eliminated has a huge impact on cycle time and the ability of data scientists and developers to quickly get their models into production. ML platforms also reduce the cognitive load on data scientists and machine learning engineers, allowing them to focus on the aspects of the ML process that are most critical while delegating everything else to the platform. Furthermore, once they've determined the right way to tackle a particular problem, a platform allows them to automate the solution so that they no longer need to worry about it.

- **Scalability.** ML platforms help teams scale in many ways. They help them scale training, decreasing the amount of time it requires to produce a performant model, and inference, allowing applications to make more predictions more consistently. More importantly though, ML platforms help teams scale their own output in terms of the number of models they are able to produce in a given time, and their capacity to deliver and take advantage of machine learning.

- **Visibility.** Platforms help teams and managers gain much-needed visibility into machine learning models in development and production. They provide a centralized resource for collecting and sharing information on the data, features, experiments, and models that these teams work with so that insights about them can be more easily gained and shared.

2. Organize for success

Almost universally, the model-driven organizations that we've interviewed have established dedicated "ML platform" or "ML infrastructure" teams to help make their data scientists and ML developers more productive.

Setting up a platform team generally happens after there is a need for multiple data science efforts to be supported simultaneously. When this occurs, ML platform teams are established to drive efficiency and ensure that both scientists and developers have ready access to the tools and resources they need to work efficiently.

Airbnb, for example, established its ML infrastructure team as demand for ML models grew across a broad set of teams. The platform team's mission is to eliminate what they call the *incidental complexity* of machine learning, as opposed to its *inherent complexity*, thus making machine learning more accessible to the company's various developers.

While the establishment of ML platform or infrastructure teams is a trend that is just beginning within enterprises, most organizations have a recent internal example to look to. In many ways, ML infrastructure teams are to data scientists and ML developers what DevOps and developer platform teams are to traditional enterprise developers. Just as the latter have become a popular way to support software developers and ensure their productivity, so will the former be to their machine learning counterparts.

3. Understand your users

Once a platform team is established, its first task is to understand its users. Each of the platform's potential users—researchers, data scientists, machine learning engineers, and software developers—will typically have different skill sets, needs, and tool preferences.

Data scientists, to generalize, will be very comfortable with statistical tools and Jupyter notebooks but are often less comfortable with tools like version control systems that are commonplace in the software development world. Conversely, software engineers will be comfortable with a variety of crude command-line tools but might benefit from a system that offers them a selection of pre-built models to choose from. Whatever the mix, talking to the user community will help the team develop a direction, establish feature priorities, and come to terms with issues like how opinionated the platform should be.

4. Consider build vs. buy

Organizations seeking to establish ML platforms for their data scientists and ML engineers should carefully consider their options before deciding to build a proprietary solution.

The "build" approach, while highly customized to the needs of the organization, is expensive and requires strong engineering talent and teams to develop and maintain the platform. The "buy" option, on the other hand, often requires adapting to a given vendor's approach but demands less time and expertise on the part of the customer.

The reality is that "build" and "buy" exist on a spectrum. While Facebook, LinkedIn, and Airbnb have each invested in dedicated engineering teams to build and maintain their own proprietary ML platforms, in the case of Facebook's FBLearner, the entire platform is largely built from scratch. At Airbnb, on the other hand, the company's platform engineering team made liberal use of existing open source tools like Jupyter, Docker, Kubernetes, Airflow, and Spark in the creation of its platform. LinkedIn's platform is arguably somewhere in the middle, based on many complex custom subsystems while taking advantage of the Hadoop and Spark ecosystems.

We believe most enterprises will ultimately compose their ML platforms from commercial, open source, or cloud-delivered software, along with custom integration and custom-coded modules as needed to address their unique needs.

5. Explore available solutions

As you may have gathered from the examples in the previous section, the machine learning tools market is growing quickly and there are many companies—startups and established vendors alike—that offer products and projects that may be of use to your organization as you build out your ML platform. So many, in fact, that the market can be quite confusing, with many of these vendors making similar and opaque claims. Understanding the market landscape will help you identify the right tools and prospective partners for your company.

Wide vs. Deep

One of the most interesting and important distinctions among the various tools available to help you build out your organization's machine learning platform is whether the tool aims to be *wide* or *deep*:

Wide/Generalist	Deep/Specialist
Wide refers to generalist tools that seek to provide end-to-end support for various aspects of the ML workflow. Wide offerings aim to give users a broad platform-in-a-box experience.	Deep refers to specialist tools that seek to solve one problem deeply. These tools typically have robust APIs and are designed to easily fit into an organization's existing ML workflow.

	Wide/Generalist	Deep/Specialist
Pros	• Easiest way to establish an ML platform; quickest path to ML platform benefits • Tightly integrated toolset requires little custom integration • Common control plane simplifies management & governance • One source for support	• Best-in-class functionality and/or performance • Flexible; coexists and easy to integrate with existing tools, workflows, and decision • Greater control over the key decisions driving platform user experience
Cons	• Individual tools may be shallow in functionality or performance • All-or-nothing; may have to abandon existing investments • Beholden to vendor's roadmap and priorities for enhancements and bug fixes • Greater risk of lock-in	• Lacks unified management & governance • Must be integrated with existing workflows/ systems • Multiple vendor relationships to manage; end-user organization shoulders greater support burden

Figure 11. Wide vs. deep tools: pros and cons

Wide vs. deep presents both buyers and vendors with an interesting paradox. Specialist tools by their nature tend to assume that users have enough of a pipeline or platform in place that the tool is easily slotted-in and can quickly demonstrate value. Often, however, specialist tool vendors find that customers have broader gaps in their workflow that prevent them from taking advantage of the tool. As a result, these companies, often small startups, find that their sales cycles are longer than anticipated as they (or the buyer) work to fill workflow gaps with custom integration.This leads specialist vendors towards expanding their offerings to take on more of the end-to-end pipeline problem so as to more quickly demonstrate value in immature customer environments.

Generalist tools, on the other hand, face a different set of challenges. First, by the time a customer is mature enough in their data science journey that they're ready to adopt an end-to-end platform, they've often already invested in building out one or more pieces of their own workflow. These customers or their users are often not too excited about needing to start from scratch with an unproven tool and throw out everything they've done. Second, the end-to-end problem is deceptively simple, but the existence of so many specialized vendors is an indication of the complexity, depth, and options inherent in many individual steps of the data science process. So the individual features of an end-to-end platform may not even be as capable, and are often not as tailored or differentiating, as the homegrown tools they seek to displace. Finally, as many vendors jump into the ring with shallow end-to-end offerings and aspirations to "own the data science pipeline," end-to-end tools become increasingly commoditized. In order to differentiate, these tools will need to build depth or specialization in one or more individual areas in order to differentiate, thus heading in the opposite direction as the specialist tools.

The result of this paradox is the increasingly confusing market in which we find ourselves today, in which everyone markets themselves as an end-to-end platform and it's up to the customer to figure out if and where any depth exists. For example, DataRobot promotes itself as an end-to-end platform for all of the modeling roles in an organization (business analyst, data scientist, software engineer, etc.). Their strength, however, is in applying AutoML methods to modest-sized tabular datasets for business analysts and "citizen data scientists." They lack many of the requirements to support large-scale custom model development in the enterprise, however, and don't offer robust support for deep learning.

Meanwhile, the open source TensorFlow Extended (TFX) project by Google is similarly marketed as an end-to-end machine learning platform. In reality, it is a collection of special-purpose modules (TensorFlow Data Validation, TensorFlow Transform, TensorFlow Model Analysis, and TensorFlow Serving) that can be used—along with other open source technologies like Apache Airflow or Kubeflow Pipelines for pipeline orchestration and Apache Beam for distributed processing—to build a custom platform for TensorFlow models and developers.

Hopefully, in this discussion, it is clear that wide vs. deep should not be equated to good vs. bad or vice versa. Rather, what's important is to realize that different technologies have different aims and that each organization will need to identify the best fit for its needs and choose accordingly.

Wide vs. deep is an important distinction, but far from the only one. Other important considerations include:

- **Target use case.** Understanding the intended use case for a given tool is key to understanding how and where to best apply it. For example, Falkonry and Reality AI both offer modeling tools targeting manufacturing and industrial use cases. The former focuses on predictive operations applications while the latter targets embedded AI systems with a wide variety of sensor connections.

- **Target user profile.** The needs and preferences of a business analyst vs. data scientist vs. ML engineer vs. platform engineer can vary widely. Different tools cater to different types of users in the decisions they make and features they offer. (See point 3, "Understand your users," above).

- **Target model type.** Platforms differ on the types of models that they target and support. For example, some platforms target traditional ML model types while others target deep learning. While they've since broadened their footprint, BigML originally only supported decision trees and random forests as model types. Some platforms are even more specific, targeting models built using a specific framework. TensorFlow Extended (TFX) is an example. Others target a specific use case. For example, Allegro and Neurala are platforms designed to help users create computer vision models.

- **Tool history/legacy.** Two similarly marketed tools can take very different approaches, often informed by the founders' backgrounds or the company's technology legacy. For example, Dotscience, a startup mentioned previously, was founded by Luke Marsden who previously founded ClusterHQ, a company focused on containerized storage systems. As a result, while their tool is marketed like other end-to-end ML platforms, its strength is in data versioning and snapshotting.

  Paperspace's end-to-end platform offering, on the other hand, grew out of an acquisition of a team whose roots were in building CI/CD systems on top of Kubernetes. As a result, their view of the world is very software engineer centric. Examining the origins of a given tool and team can help with understanding the tool's best application and use-case.

- **Open vs. closed source vs. SaaS vs. cloud.** ML platforms are delivered in a wide variety of formats, including open and closed source software that teams can run in their own datacenters, software that is supported running in any cloud, and SaaS software, whether supplied by one of the large cloud vendors or an independent firm.

6. Learn from DevOps efforts

We've alluded to this point earlier in this ebook, but it's worth reiterating here. Our efforts to industrialize and scale machine learning mirror in many ways the parallel evolution that has taken place in software development over the past decade and much can be learned from both the process and the result.

Modern software development practices emphasize strong problem definition (user stories), tight iterative loops (sprints), high degrees of automation (CI/CD), high levels of repeatability (Docker containers), and robust platforms that provide developer services (PaaS) and manage the underlying infrastructure (container orchestration, Kubernetes). Of course, the analogy isn't perfect and can be taken too far. Still, your organization likely has learned a lesson or two about providing platform services to teams of developers, and these lessons can be applied to supporting your data scientists as well.

7. Start small

It would be hard to overemphasize the evolutionary nature of ML platform development and deployment. None of the organizations we've profiled here, or any of the others that we've talked to, have deployed their machine learning platform in a single “big bang.” Rather, each organization's ML platform evolved in a unique way based on its needs, users, skill sets, organizational structure, and existing technology investments.

In this ebook we've identified a broad set of capabilities to be considered for supporting and accelerating machine learning in the enterprise. Your team does not need to build an entire end-to-end system for your efforts to be successful. Rather, teams should prioritize their efforts based on a careful analysis of your users' specific needs and your organization's ability to execute. Talking to your organization's data scientists and ML engineers will likely yield a short-list of pain points that could be alleviated with off-the-shelf or custom-developed tools.

If your organization is new to data science, or you're looking for general guidance on making data scientists more effective, we recommend reviewing Monica Rogati's, *Data Science Hierarchy of Needs*[46] below:

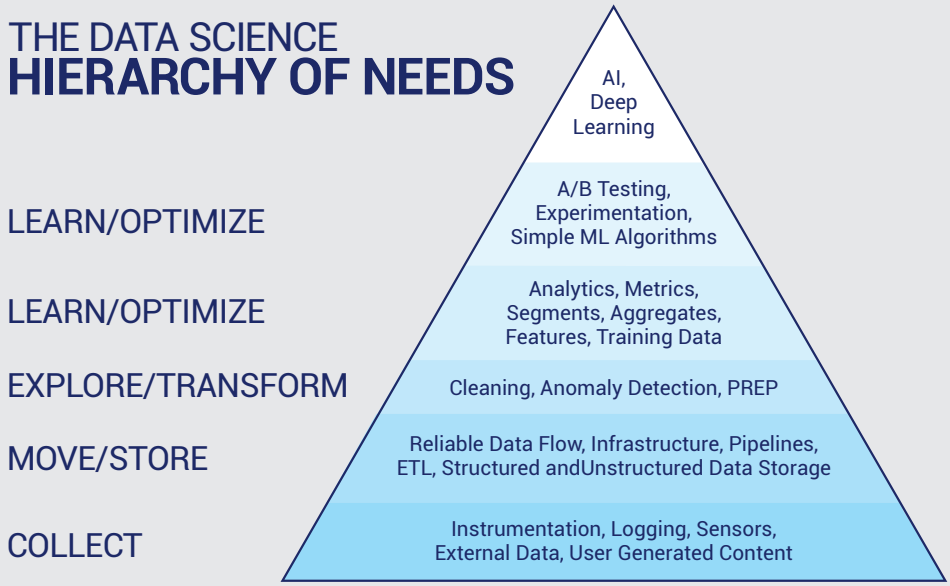

Figure 12. Data Science Hierarchy of Needs

Conclusions

We began this ebook with a discussion of the increasing importance of machine learning models to the enterprise. If you agree that AI, and ML models in particular, will help drive enterprise competitiveness in the future, then the premise of this ebook will not produce much of an argument: enterprises will need to put people, processes, and technologies in place that allow them to deliver ML models into production more efficiently, consistently, and scalably.

With this future as a backdrop, the importance of ML platforms becomes clear. Over the next few years, enterprises will deploy ML platforms in increasing numbers because they help make the benefits of machine learning more readily accessible at greater scale.

The ML platforms landscape is evolving rapidly. Awareness of the importance of ML platforms and the availability of solutions in this area has grown dramatically in the past year. We hope we've had a small part to play in this via our coverage of the topic on the TWIML AI Podcast.

To further support the development of an informed, sustainable community of technologists equipped to meet the current and future ML platforms and infrastructure needs of their organizations, we're excited to host **TWIMLcon: AI Platforms**. This is the first conference of its kind to focus on practical ML and AI platforms and infrastructure. We are dedicated to creating a forum where data scientists, ML engineers, and platforms and infrastructure

practitioners, leaders, and innovators can share, learn, and connect with one another on the platforms, tools, technologies, and practices necessary to industrialize and scale the delivery of machine learning and AI in the enterprise.

We invite you to join us for two exciting days of practical presentations, lively discussion, and great community interactions. Learn more and register at twimlcon.com.

Needless to say, our research in and coverage of this topic will continue. Be sure to visit twimlai.com/platforms to access additional content and resources in this area, including profiles of the vendors, products, and projects mentioned in this ebook. While you're there you can also join our growing community of ML innovators working in this area.

Hope to see you soon!

References

1. Carlos A. Gomez-Uribe and Neil Hunt, "The Netflix Recommender System: Algorithms, Business, Value, an Innovation," ACM Transactions on Management Information Systems, January 2016, https://dl.acm.org/citation.cfm?id=2843948.
2. Sam Charrington, "Systems and Software for Machine Learning at Scale with Jeff Dean," This Week in Machine Learning & AI, April 2, 2018, https://twimlai.com/twiml-talk-124-systems-software-machine-learning-scale-jeff-dean/.
3. Sam Charrington, "TWIML Presents: AI Platforms Volume 1," This Week in Machine Learning & AI, November 2018, https://twimlai.com/aiplatforms2018/.
4. Sam Charrington, "TWIML Presents: AI Platforms Volume 2," This Week in Machine Learning & AI, May 2019, https://twimlai.com/aiplatforms2/.
5. Robert Chang, "Using Machine Learning to Predict Value of Home on Airbnb," Medium, July 17, 2017, https://medium.com/airbnb-engineering/using-machine-learning-to-predict-value-of-homes-on-airbnb-9272d3d4739d.
6. Andrew Hoh and Nikhil Simha, "Zipline: Airbnb's Machine Learning Data Management Platform," SAIS 2018, June 12, 2018, https://databricks.com/session/zipline-airbnbs-machine-learning-data-management-platform.
7. Jeffrey Dunn, "Introducing FBLearner Flow: Facebook's AI Backbone," Facebook Engineering, May 9, 2016, https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone.
8. Kim Hazelwood, et al, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," Facebook, Inc., February 24, 2018, https://research.fb.com/wp-content/uploads/2017/12/hpca-2018-facebook.pdf.
9. Sam Charrington, "Holistic Optimization of the LinkedIn Newsfeed with Tim Jurka," This Week in Machine Learning & AI, January 28, 2019, https://twimlai.com/twiml-talk-224-holistic-optimization-of-the-linkedin-newsfeed-with-tim-jurka/.
10. Sam Charrington, "Scaling Machine Learning on Graphs at LinkedIn with Hema Raghavan and Scott Meyer," This Week in Machine Learning & AI, March 4, 2019, https://twimlai.com/twiml-talk-236-scaling-machine-learning-on-graphs-at-linkedin-with-hema-raghavan-and-scott-meyer/.
11. Sam Charrington, "End-to-End Data Science to Drive Business Decisions at LinkedIn with Burcu Baran," This Week in Machine Learning & AI, April 24, 2019, https://twimlai.com/twiml-talk-256-end-to-end-data-science-to-drive-business-decisions-at-linkedin-with-burcu-baran/.
12. Sam Charrington, "Productive Machine Learning at LinkedIn with Bee-Chung Chen," This Week in Machine Learning & AI, November 15, 2018, https://twimlai.com/twiml-talk-200-productive-machine-learning-at-linkedin-with-bee-chung-chen/.
13. Jermey Hermann and Mike Del Balso, "Meet Michelangelo: Uber's Machine Learning Platform," Uber Engineering, September 5, 2017, https://eng.uber.com/michelangelo/.

References - continued

14. Tim Sell and Willem Pienaar, "Introducing Feast: an open source feature store for machine learning," Google Cloud, January 18, 2019, https://cloud.google.com/blog/products/ai-machine-learning/introducing-feast-an-open-source-feature-store-for-machine-learning.
15. Logical Clocks, "Hopsworks: Full-Stack Platform for Scale-Out Data Science," Github, https://github.com/logicalclocks/hopsworks.
16. Ben Mabey, "Provenance: Provenance and Caching Library for Python Functions," Github, https://github.com/bmabey/provenance.
17. Robert Chang, "A Beginner's Guide to Data Engineering — The Series Finale," Medium, June 24, 2018, https://medium.com/@rchang/a-beginners-guide-to-data-engineering-the-series-finale-2cc92ff14b0.
18. SigOpt is a sponsor of this ebook.
19. IDSIA, "Sacred: Infrastructure for Computational Research", Github, https://github.com/IDSIA/sacred.
20. Google, "TensorBoard: TensorFlow's Visualization Toolkit," Tensorflow, https://www.tensorflow.org/tensorboard.
21. Manasi Vartak, et al, "ModelDB: A System to Manage Machine Learning Models," Github, https://github.com/mitdbg/modeldb.
22. Kubeflow, "Kubeflow: Machine Learning Toolkit for Kubernetes," Github, https://github.com/kubeflow/.
23. Sam Charrington, "Kubernetes for Machine Learning, Deep Learning & AI," This Week in Machine Learning & AI, https://twimlai.com/kubernetes/.
24. James Kanter and Kalyan Veeramachaneni, "Deep Feature Synthesis: Towards Automating Data Science Endeavors," 2015, http://www.jmaxkanter.com/static/papers/DSAA_DSM_2015.pdf.
25. Feature Labs, "Featuretools: An Open Source Python Framework for Automated Feature Engineering," Github, https://github.com/featuretools/featuretools.
26. Wikipedia contributors, "Hyperparameter optimization," Wikipedia, The Free Encyclopedia, August 13, 2019, https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=910615943.
27. Yelp, "MOE: A Global, Black Box Optimization Engine for Real World Metric Optimization," Github, February 23, 2014, https://github.com/Yelp/MOE.
28. Frank Hutter, et al, "SMAC," AutoML Freiburg-Hannover, https://www.automl.org/automated-algorithm-design/algorithm-configuration/smac/
29. Scikit-Optimize, "Scikit-Optimize: Sequential Model-based Optimization with a `scipy.optimize` Interface," Github, https://github.com/scikit-optimize/scikit-optimize.
30. Kubeflow, "Katib: Kubernetes Native System for Hyperparameter Tuning and Neural Architecture Search," Github, April 1, 2018, https://github.com/kubeflow/katib.

References - continued

31. Ruben Martinez-Cantin, "BayesOpt: A Toolbox for Bayesian Optimization, Experimental Design and Stochastic Bandits," Github https://github.com/rmcantin/bayesopt.
32. Hyperopt, "Hyperopt: Distributed Asynchronous Hyperparameter Optimization in Python" Github, Septermber 4, 2011, https://github.com/hyperopt/hyperopt.
33. UCBerkeley RISELab, "Tune: Scalable Hyperparameter Search," Github https://github.com/ray-project/ray/tree/master/python/ray/tune.
34. Cade Metz, "Building A.I. That Can Build A.I.," The New York Times, November 5, 2017, https://www.nytimes.com/2017/11/05/technology/machine-learning-artificial-intelligence-ai.html
35. Google, "Cloud AutoML," Google Cloud, https://cloud.google.com/automl/.
36. Kubeflow, "Introduction to Kubeflow Pipelines," Kubeflow, https://www.kubeflow.org/docs/components/pipelines/pipelines/.
37. Nteract, "Papermill: Parameterize, Execute, and Analyze Notebooks," Github, https://github.com/nteract/papermill.
38. Sam Charrington, "Bighead: Airbnb's Machine Learning Platform with Atul Kale," This Week in Machine Learning & AI, November 8, 2018, https://twimlai.com/twiml-talk-198-bighead-airbnbs-machine-learning-platform-with-atul-kale/.
39. Sam Charrington, "Supporting TensorFlow at Airbnb with Alfredo Luque," This Week in Machine Learning & AI, March 29, 2019, https://twimlai.com/twiml-talk-244-supporting-tensorflow-at-airbnb-with-alfredo-luque/.
40. Sam Charrington, "Productizing ML at Scale at Twitter with Yi Zhuang," This Week in Machine Learning & AI, June 3, 2019, https://twimlai.com/twiml-talk-271-productizing-ml-at-scale-at-twitter-with-yi-zhaung/.
41. IBM, "Fabric for Deep Learning," Github, https://github.com/IBM/FfDL.
42. Neoklis Polyzotis, et al, "Data Management Challenges in Production Machine Learning," Google, 2017, https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46178.pdf.
43. Google, "TensorFlow Serving: Architecture," TensorFlow, https://www.tensorflow.org/tfx/serving/architecture.
44. Christopher Olston, et al, "TensorFlow-Serving: Flexible, High-Performance ML Serving," NIPS 2017, http://learningsys.org/nips17/assets/papers/paper_1.pdf.
45. Julien Simon, "Mastering the Mystical Art of Model Deployment," Medium, July 28, 2018, https://medium.com/faun/mastering-the-mystical-art-of-model-deployment-c0cafe011175.
46. Monica Rogati, "The AI Hierarchy of Needs," Hackernoon, June 12, 2017, https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007.

About This Week in Machine Learning & AI

Machine learning and artificial intelligence are dramatically changing how businesses operate and people live. Through our podcast, publications, and community, This Week in Machine Learning & AI brings top minds and ideas from the world of AI to a broad and influential community of data scientists, engineers and tech-savvy business and IT leaders.

TWiML was created and is hosted by Sam Charrington, a sought after speaker, commentator and thought leader, and the founder of CloudPulse Strategies, an industry analysis and consulting firm. Sam's professional interests center on the many enterprise applications of machine learning and AI, bringing AI-powered products to market, and AI-enabled and -enabling technology platforms.

CloudPulse Strategies creates and curates intelligent content that helps makers make and executives execute, providing an inside look at the real-world application of "intelligent platform" technologies. We build and support communities of innovators who are as excited about these technologies as we are. And we advise organizations to help craft strategies for taking advantage of the vast opportunities created by ML and AI.

Connect with us:

- sam@twimlai.com
- @samcharrington
- @twimlai
- www.linkedin.com/in/samcharrington
- twimlai.com