AI Infrastructure Alliance

2022

AI Infrastructure Ecosystem

Version 20220721001

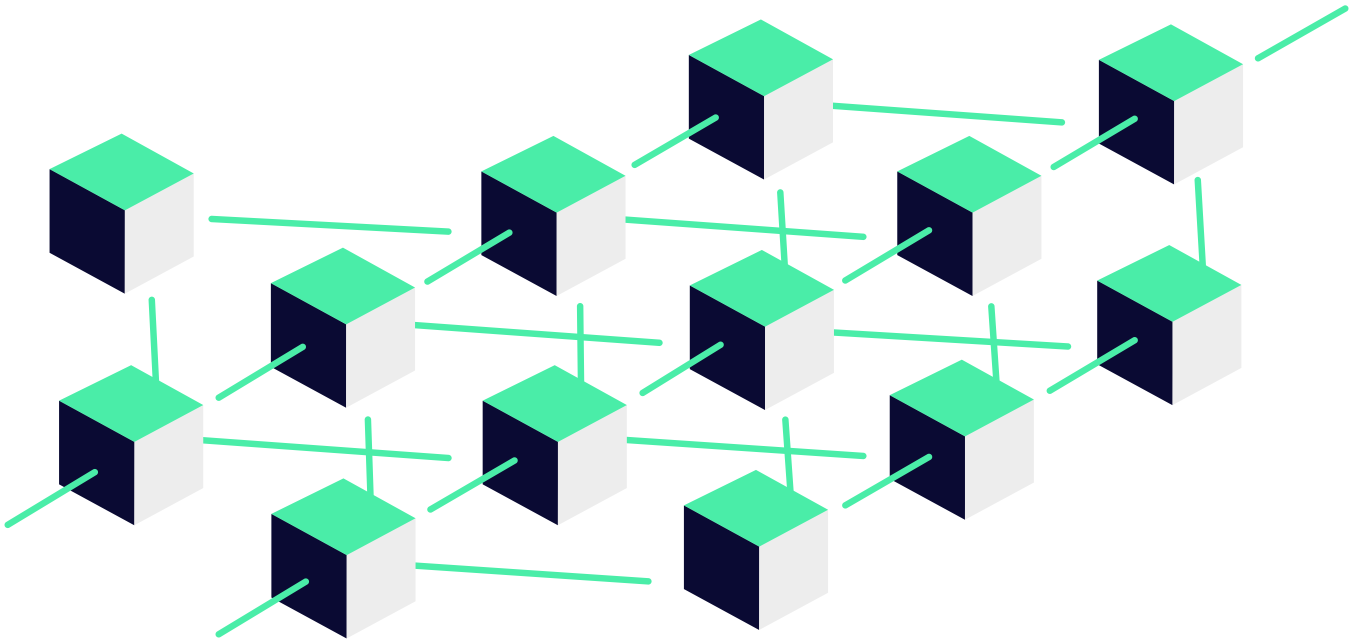


Table of Contents

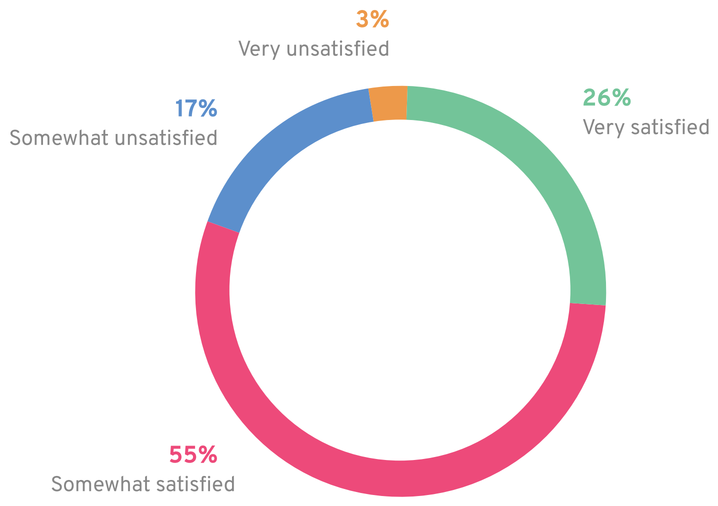
MANAGEMENT SUMMARY

Intelligence is the great competitive advantage in history. It's not just whether we have it, but whether we can wield it effectively. But over the last decade, intelligence has undergone a profound change. It's no longer just confined to our heads because the rise of artificial intelligence and machine learning (AI/ML) techniques has now made it possible to productionize intelligence to solve previously unsolvable challenges. We can use AI to spot problems early on in production lines, predict customer churn, reclaim budgets, streamline support requests, translate languages, highlight key passages in legal documents, detect fraud, iterate on new design ideas and much, much more.

Yet this production intelligence has largely remained the province of highly technical teams and big tech companies. Often these teams built their own AI/ML infrastructure from scratch because there was nothing on the market to support their efforts. Yet over the last five years, we have seen a rapid proliferation of new tools and platforms that allow enterprises and small to medium businesses to benefit from the intelligence revolution. However, building the right AI/ML infrastructure that fits specific company needs is still a significant challenge.

Only 26% of teams we surveyed were very satisfied with their current AI/ML infrastructure. Fifty-five percent were only somewhat satisfied, while 17% were somewhat unsatisfied, and 3% were very unsatisfied. In other words, most teams see a lot of room for improvement.

How satisfied are you with the current state of your AI/ML infrastructure?

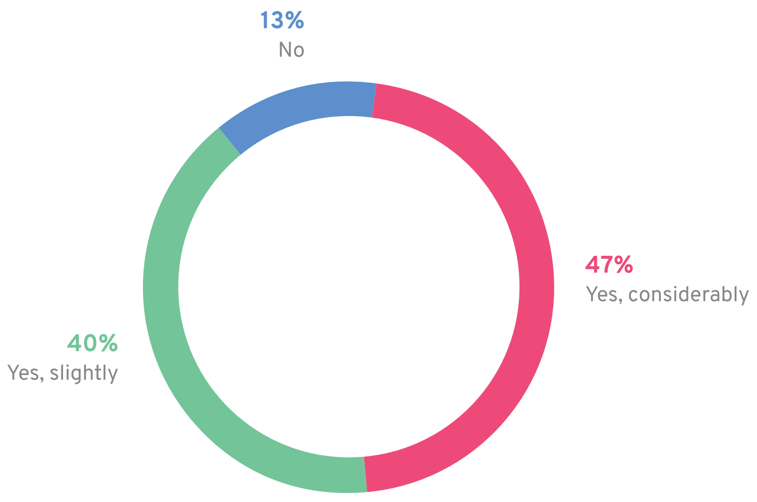


This is because the AI/ML infrastructure landscape is vast, complex and rapidly evolving. It's difficult to understand the capabilities of each platform and to see where they fit into existing systems without a lot of research and time invested. AI/ML systems are also complex because there is no one tool that does everything perfectly, so building a modern AI/ML stack involves many different tools and components. Even worse, marketing teams often obscure the capabilities of systems or promise that a platform can do everything equally well, when the reality is usually very different. Finally, building a robust infrastructure requires buy-in from many different stakeholders across an organization, everyone from data scientists to data engineers to IT infrastructure architects to support teams to network and security engineers.

Up to this point, much of what has been written on AI/ML focuses on building excitement around AI, describing the state of ML adoption, or outlining the state of AI/ML research. While it is nice to read about the incredible possibilities that come from utilizing AI/ML in your company, what is truly needed is thorough coverage of the infrastructure available and the possible directions your company can take to achieve your business goals. Here we focus on how to sustainably build your AI/ML infrastructure to set your company up for success over the long term, as your capabilities, needs and demands evolve.

AI/ML teams are growing and this report aims to give every company the keys to build their AI/ML infrastructure by providing a comprehensive and clear overview of the AI/ML infrastructure landscape.

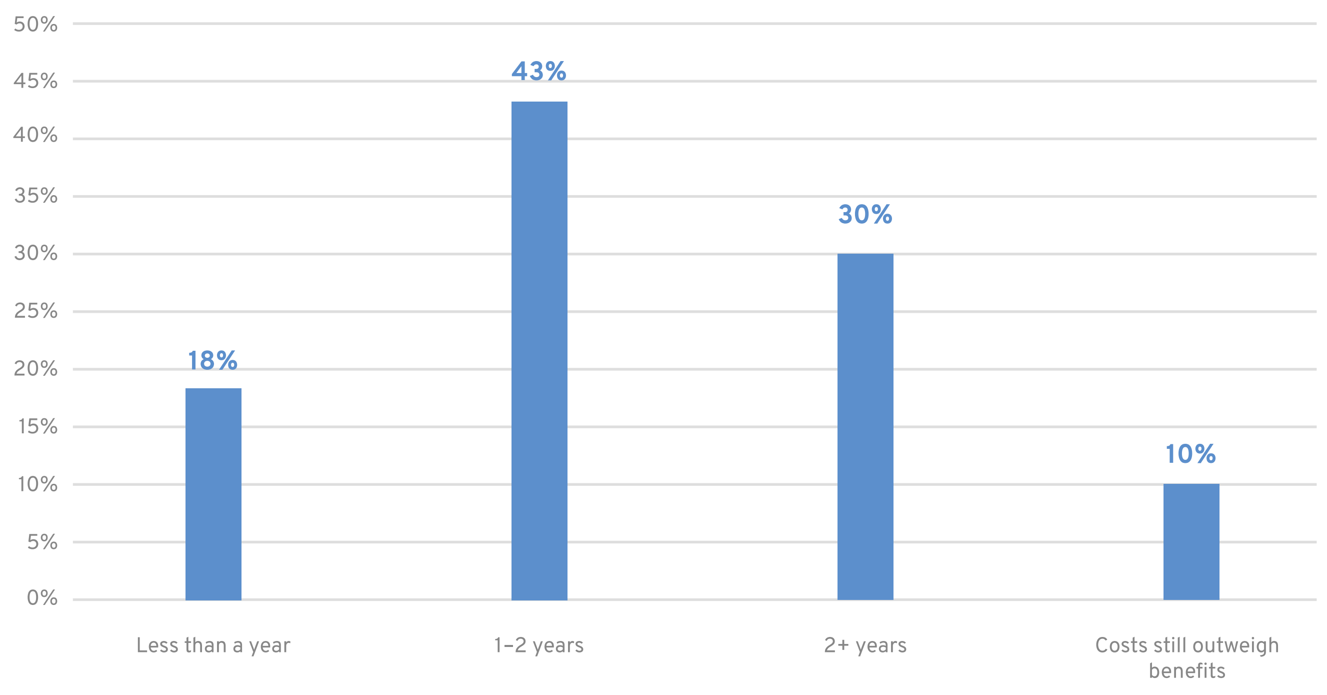
Has your AI/ML team grown over the past year?



We provide insights on realistic capabilities and tradeoffs for many different platforms, as well as projections about how infrastructure requirements may evolve over the next five years according to AI/ML experts.

The good news is that the majority of companies we surveyed found that the benefits they got from their AI/ML infrastructure outweighed the costs in two years or less. That means if you invest in the right infrastructure, after surveying the field and considering it carefully, you can reap rewards swiftly.

How long did it take for the benefits you get from AI/ML to outweigh the costs of infrastructure, implementation and personnel?



By aligning your business goals with your wisely built AI/ML infrastructure, you can push the boundaries of what is possible for your company now and in the future.

INTRODUCTION

Over the past five years, we've seen a massive surge in the use of artificial intelligence (AI) and machine learning (ML) across every industry, as well as a stream of reports that cover the adoption rate of ML in enterprises and reports on the state of AI/ML research from prominent academic and research institutes like Stanford.

This report has a distinctly different focus. Here we deliver a comprehensive overview of the state of AI/ML **infrastructure** software, the software that powers the building, deploying, scaling and running of AI/ML models.

The target audience for this report is any executive or engineering team lead looking to find the right tools to establish or expand their in-house AI/ML practice. It will show you how the field has developed, as well as what the key capabilities of various AI infrastructure categories are now and where they are going over the next five years. By the end, you will have a clear vision of where to invest your budget to accelerate your projects and take them to the next level.

We describe the various categories according to the AI Infrastructure Alliance's (AIIA) MLOps blueprint and then discuss each category in its own section. If you are unsure what category would make the biggest impact, then you can read the report from start to finish to get a better idea of what parts of the stack are essential to building or expanding your team's capabilities. Alternately, if you are already clear on what kinds of tools you need next, then you can skip to the section that covers your needs straightaway. However, you may still wish to read the rest of the report as there may be capabilities in the space that you aren't aware of yet and that may prove valuable to your efforts.

The Three Approaches to AI/ML Platforms

There are three primary approaches to building an AI/ML platform.

1. Build your own
2. Buy an end-to-end solution
3. Best of breed

These are roughly the same approaches that organizations need to consider when it comes to building a web application platform or an in-house IT system, but there are some wrinkles when it comes to AI/ML that make it different. The most notable difference is the maturity of the space. Because enterprise AI/ML is a relatively young field, there is a vast array of products and services. In some categories of the space, there are clear market leaders, and in others there is a range of potential choices that could make sense depending on your organizational needs.

This wider range of platforms mirrors earlier technological advances. There were hundreds of car companies in the early days of the automobile industry, many of them small shops producing a few cars a year, before the assembly-line processes pioneered by Ford took over. In the early days of the web, there were no fewer than fifty different web servers, but most IT administrators would be hard pressed to think of more than three or four today, with Apache, NGINX, IIS and few other servers dominating the marketplace.

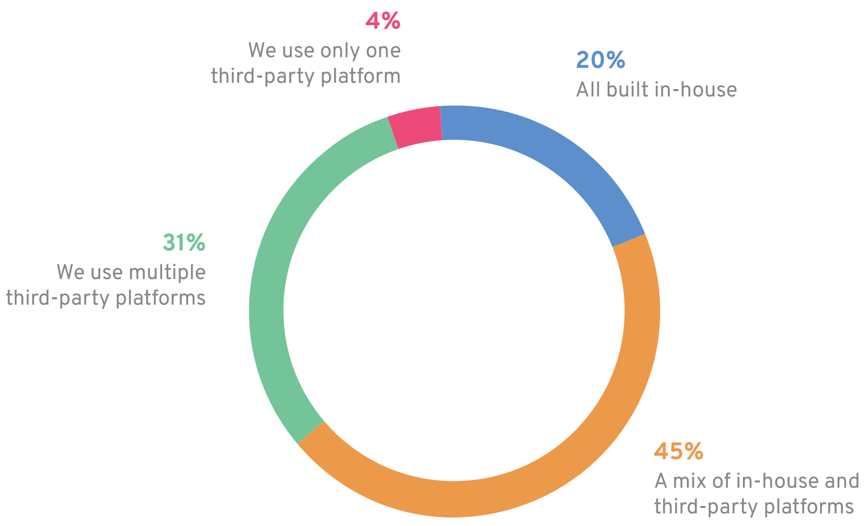
Build Your Own

The first companies to adopt AI/ML techniques at scale were largely tech companies like Google, Tesla, OpenAI, DeepMind and Netflix. Because the field was new, they had little choice but to build their own solutions for building, training, deploying and running AI/ML models in production. But as those pioneers blazed new territory and AI/ML techniques came out of the labs, universities and big-tech companies, enterprise software companies and startups emerged to fulfill the demand for enterprises who generally don't have the in-house expertise or resources to build an entire bespoke IT system from scratch and then maintain it and updated it.

The AIIA does not recommend that most companies attempt to build their own AI/ML platform from scratch or that they attempt to stitch one together from pure open-source components alone. This approach is incredibly complex and prone to failure. It is useful only for highly advanced teams, with very specific and idiosyncratic use-case requirements that are not met by current software platforms. Even if you suspect that your needs are unique, we highly recommend you put together a team to study whether that is truly the case or whether you would be better served by a set of tools already developed.

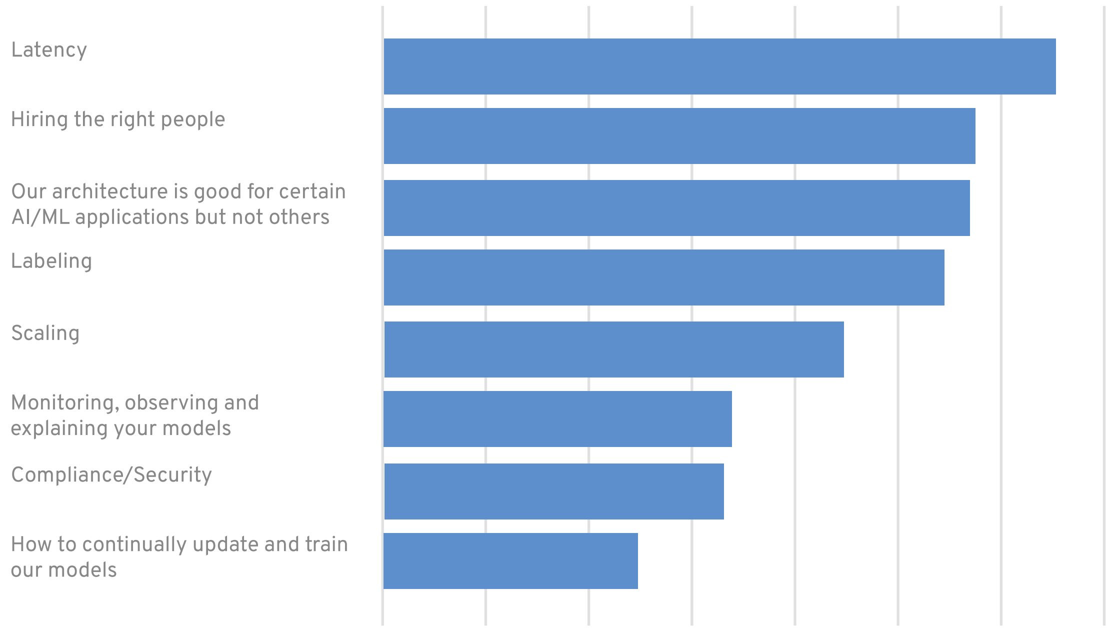
The good news is that, outside of very specialized circumstances, most companies are not building their own architecture from scratch anymore. According to our enterprise survey, only 20% of companies built their entire infrastructure in-house, while 45% use a mix of in-house and third-party tools, and 31% use a mix of third-party tools exclusively.

How do you categorize your AI/ML infrastructure? (All)



While early pioneers had no choice but to build their own infrastructure, increasingly we're seeing companies choose a mixture of buying and building or simply buying as more and more robust products make it to market. This is because long-term maintenance of custom-built applications requires tremendous engineering resources. In addition, teams often discover too late that the platform they expected to do everything proved too brittle and rigid for use cases beyond the initial design. In fact, that was the third-biggest challenge teams told us they faced when building their AI/ML infrastructure: the platform turned out to be only good for certain applications and not for others.

What are the most significant challenges you face when building your AI/ML infrastructure?



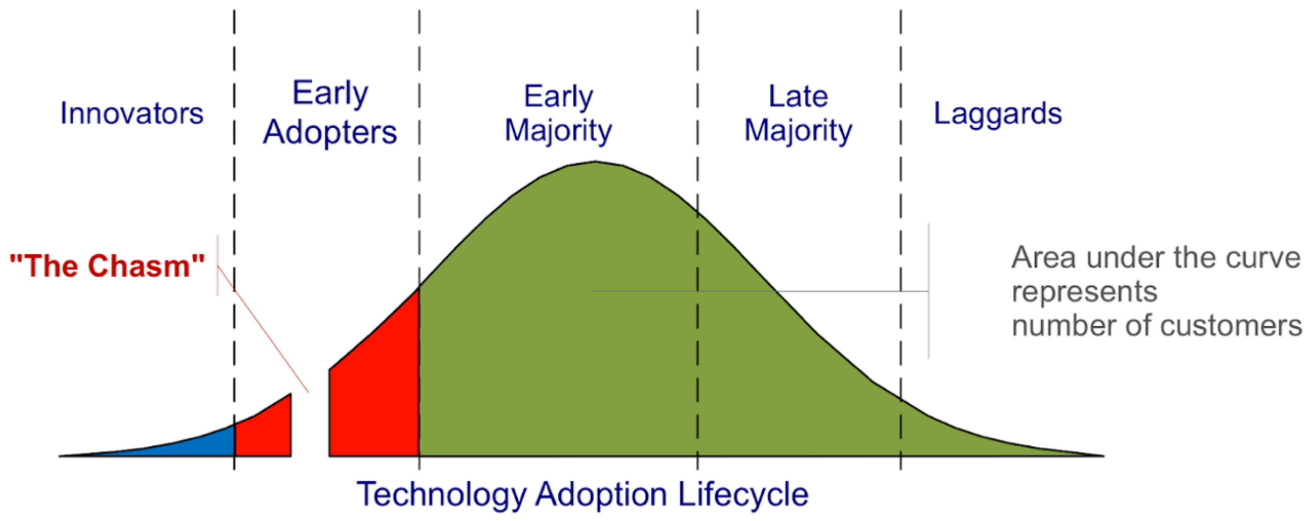
That was the case with the Michelangelo platform at Uber, headed by Mike Del Blaso, who later left Uber to start Tecton, a feature store platform. While the platform proved highly useful for UberEats when they built it, Del Blaso later noted that the platform proved very effective for the few use cases they built it for, but it was not easily generalizable to other use cases. The Uber team moved on to newer platforms, and at the time of this writing, many of the original components of Michelangelo have fractured into smaller, more nimble projects on the the Uber open-source Github.

That said, we do expect advanced teams to have some bespoke parts of their platform, including but not limited to glue code, custom overarching workflow interfaces across platforms and in-house built programs, libraries and frameworks. This trend is expected to persist over the next five years, as software platforms evolve to meet the vast majority of AI/ML lifecycle needs, because during that evolution there will always be gaps that need to be filled.

End-to-End

The second solution is to buy a single, unified end-to-end AI/ML platform to serve all your machine-learning and analytics needs. These are platforms that attempt to cover every single aspect of the AI/ML lifecycle. Buying a single solution that handles the vast majority of an organization's needs is a well-known and trusted buying practice in the industry, and so it is tempting to take the same approach to buying an AI/ML platform. For instance, many organizations standardized by using a single database, such as Oracle, in the mid-2000s or the VMware suite for virtualization in the 2010s, before the rise of public clouds largely displaced VMware as the dominant force in enterprise data centers.

However, the AIIA recommends that organizations resist the siren song of a unified, end-to-end solution for AI/ML platforms for several reasons. The first is that we are still largely in the early adopter phase of the AI-infrastructure ecosystem. In his 1962 book Diffusion of Innovation, sociologist Everett Rogers showed us that people and enterprises fall into five distinct groups when it comes to taking on new tech. Geoffrey Moore built on these ideas in his business bestseller Crossing the Chasm.


Innovators
Early Adopters
Early Majority
Late Majority
Laggards
"The Chasm"
Area under the curve represents number of customers
Technology Adoption Lifecycle

(Source: "Technology Adoption Lifecycle" by Craig Chelius, licensed under CC BY 3.0.)

As is typical of the development in this phase of a technological cycle, there is no single solution that meets every need and has surpassed all of its competitors in every aspect of building, training, deploying, securing and managing models in production. Instead there are a number of platforms in rapid evolution that will likely expand their capabilities over time, as well as consolidate and merge. Highly developed platforms that cover a broad range of needs tend to develop and solidify late in the early majority to late majority stage of the technological development lifecycle.

However, this has not stopped many marketing teams from declaring their solution the one solution to rule them all. We recommend that organizations cast a wary eye on vendor marketing. We also recommend that when considering a purchase, organizations look very carefully at the capabilities outlined in this report and then ask serious questions of any vendor that claims to support every aspect of the AI/ML lifecycle. As of the date of this report's writing, we have found no single solution that legitimately covers the vast array of innovation happening in the space currently, which includes work on robust ingestion and storage of structured and unstructured data, data versioning and lineage, synthetic data generators, feature stores, model registries, highly scalable pipelining and orchestration systems, deployment systems, and highly scalable serving engines, as well as state-of-the-art monitoring, explainability and observability.

That said, it is entirely possible to select one or two vendors or platforms as the core of your AI/ML platforms and then build around those cores. Many platforms detailed in this report do cover a wide range of capabilities that would serve the needs of complex enterprises. It is up to each organization to deeply understand their own needs, now and in the future, before buying their next-generation platform. For instance, if your use cases are largely structured and semi-structured today, with a focus on classic analytics tasks like customer demand, churn prediction and fraud detection, then you may standardize something like Spark as your processing engine. However, you may discover later that you've moved into deep learning and unstructured use cases like video analytics only to discover that Spark is not ideal for those workloads. At the same time, you may get a limited number of monitoring capabilities from your core vendor, but not get the full range of monitoring, observability and explainability tools offered by a vendor dedicated to those capabilities.

Lastly, it's worth noting that while cloud vendors offer solutions that seem more end-to-end, if we look a little closer, we often find that's not the case. Amazon's SageMaker is a suite of tools that offers everything from data wrangling to pipelines. However, often those tools are not well integrated, and they exist as standalone tools in the suite, much as if you'd bought a series of tools yourself. They're also highly focused on particular use cases, such as structured data, and don't handle as easily unstructured data use cases like video, images, audio and free-form text. In addition, many of these tools are tools that cloud vendors developed themselves instead of adopting a well-known, industry leading platform. This is often because a leader hasn't emerged because we're still on the early part of the adoption curve.

That also means that in the long run, as tools become more widespread, cloud vendors are likely to swap out big pieces of their suite for alternatives that have gained more traction, which raises the question: why not start with the best of breed in the first place? It's unlikely that in the long run Amazon's feature store will end up the standard feature store versus a dedicated open-source solution like Feast or a commercial platform like Molecula or Tecton, all of which run on multiple clouds. Public clouds are best at commoditizing components that have already developed much further along the technology adoption curve. Because we are still at an early point on the adoption curve, it's best to evaluate any public cloud AI/ML solution as no different from any other vendor's solution in terms of capabilities and not expect them to have already delivered a comprehensive, unified, end-to-end solution despite what their marketing might promise.

Best of Breed

The AIIA advocates the best-of-breed approach for medium to advanced data science and data engineering teams. That means taking a modular approach to building an AI/ML stack. Organizations should look to evaluate and select the leaders in different categories or consider the special capabilities of a specific vendor in the category that meets a need unique to their use case.

We recommend that you choose one or two core platforms that meet a wide variety of your needs, including data processing, pipelining to versioning and lineage, experiment tracking and deployment. After you've selected a core platform, you can more easily choose satellite platforms that meet more specific needs, such as synthetic data, feature stores, or monitoring, observability and explainability.

At this point in the evolution of AI/ML systems, the best-of-breed solution will likely demand some integration work on your team, so ensure that the platforms you choose have clean, well-documented APIs, as well as simple points of ingress and egress into and out of them. If your core platform already has integrations with other platforms you're looking to adopt, that's especially promising, but be sure to investigate the depths of those integrations. Are the integrations loose, well developed or tightly integrated at multiple levels?

To choose a core platform, carefully evaluate all of your current machine-learning use cases for the current moment, the next year and the next five years. Ensure that you completely understand the kinds of problems you're looking to solve now and over your future timeline. Because the state-of-the-art technology in AI/ML and the infrastructure that supports it are rapidly advancing, the AIIA believes that it's difficult to know beyond a shadow of a doubt whether you can support all the use cases that will develop beyond five years, and so it is best to focus on a platform that can support the vast majority of your workloads over that time horizon.

However, even though you may not be able to predict every possible use case beyond that timeframe, give yourself room to expand into other use cases you might not imagine at the moment. That means choosing a platform with maximum flexibility, language agnosticism and the ability to process structured, semi-structured and unstructured data. Ensure that your core platforms have the ability to meet all of those potential use cases.

While your team may be starting with low-hanging fruit use cases like churn prediction and customer-demand forecasting, it's not enough to select a core tool that only meets those needs, especially if you see the possibility of more advanced use cases like computer vision, audio transcription, NLP and more. Your core platform should be flexible enough to handle a wide variety of use cases.

When evaluating each platform, don't simply accept the marketing copy that claims a product can do anything and everything. Look to see well-documented use cases and examples that cover each and every aspect of what you hope to achieve now and in the future.

A mistake in choosing a core platform is one of the most costly mistakes a team can make. You may find yourself using a second or even third platform to accomplish tasks because the original core platform claimed to meet your needs in their marketing, but the reality proved very different. Mistakes in choosing the satellite platforms that support your core platform are more correctable. It's easier to swap out a monitoring platform if it doesn't meet your needs versus swapping out your primary pipeline and orchestration system.

Of course, while a best-of-breed approach provides the best chance of success in the current AI/ML infrastructure landscape, there are several downsides that must be considered. The first is cost. There is a cost associated with buying multiple platforms, and that will have to be weighed against the cost of developing, upgrading and supporting an in-house platform or being forced to add on to an "end-to-end" solution that didn't end up being end-to-end. The cost will also depend on whether you have a rich and varied set of use cases and whether you expect your team and use cases to expand over the next five years. The second challenge is support. You will not have "one throat to choke" when it comes to support and will need to deal with multiple teams. However, in the modern enterprise, we find that teams are already used to juggling multiple support contracts, and this is generally not considered as big of a barrier as it once was a decade ago.

Despite those two caveats, the AIIA still considers the modular, best-of-breed approach to be the most effective way to build an AI/ML stack today that will meet a wide variety of needs now and tomorrow, while delivering the most flexibility and ROI to your team.

THE BLUEPRINT AND THE LANDSCAPE

The major challenge any organization faces when trying to choose an AI/ML platform or set of platforms to meet their needs is how to categorize the capabilities of those platforms. What's needed are a clear set of categories and capabilities that fit each category. They should match the reality of the features of the platforms that are available now.

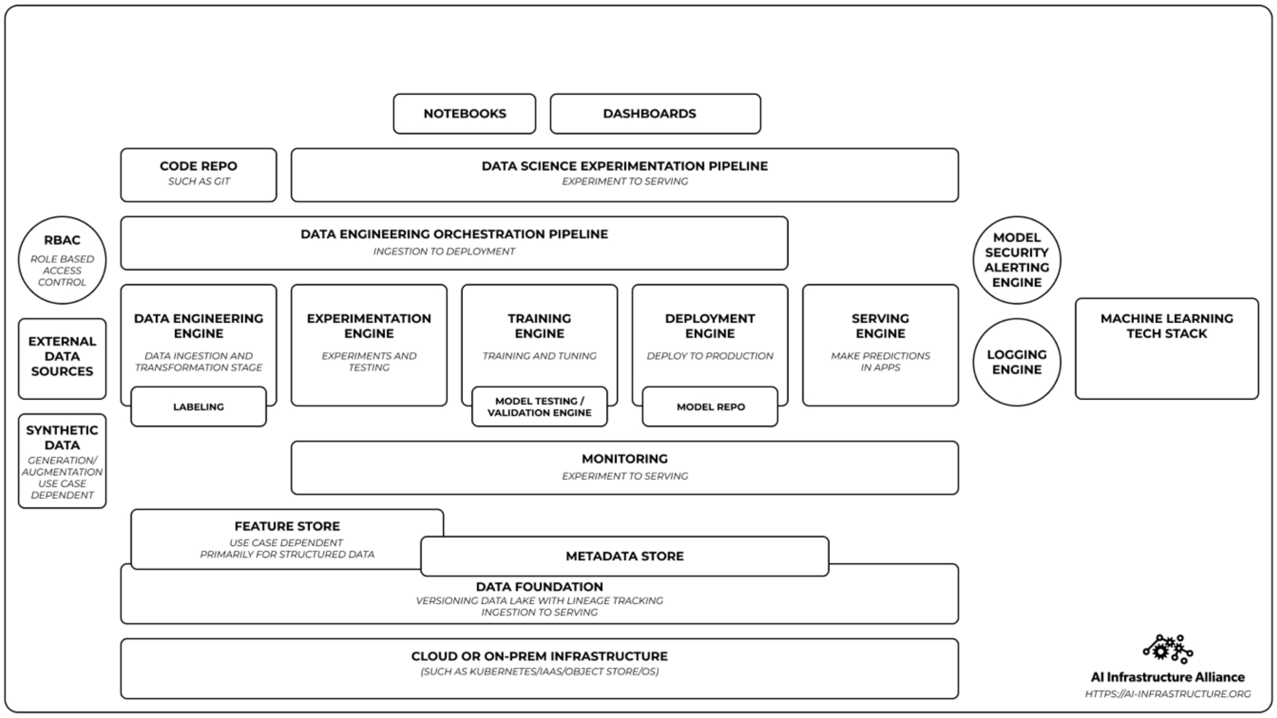
It seems simple, but it has proven challenging because the space is evolving so quickly and it exists on a new branch of the software development tree. There are a number of overlaps with traditional software development, but there are some striking differences as well. For instance, there is no analogous step to training in hand-coded software development.

A number of teams and organizations have tried to help create those categories, but by and large they've made the problem worse. Almost inevitably, the categories are poorly thought out, ill-defined and overlapping. Unfortunately, marketing largely adds to the confusion by creating an all-too-familiar landscape guide with those poorly chosen categories and then slotting each vendor's logo into a neat little box. It creates a slick graphic that is shareable on social media but is utterly ineffective at helping people understand the AI/ML software landscape. The biggest problem with this kind of graphic is that software often does not fit neatly into a single category. A platform might have labeling capabilities, experiment tracking capabilities, a feature store and more. Putting that platform's logo in the experiment tracking category alone is reductionist at best and outright wrong at worst.

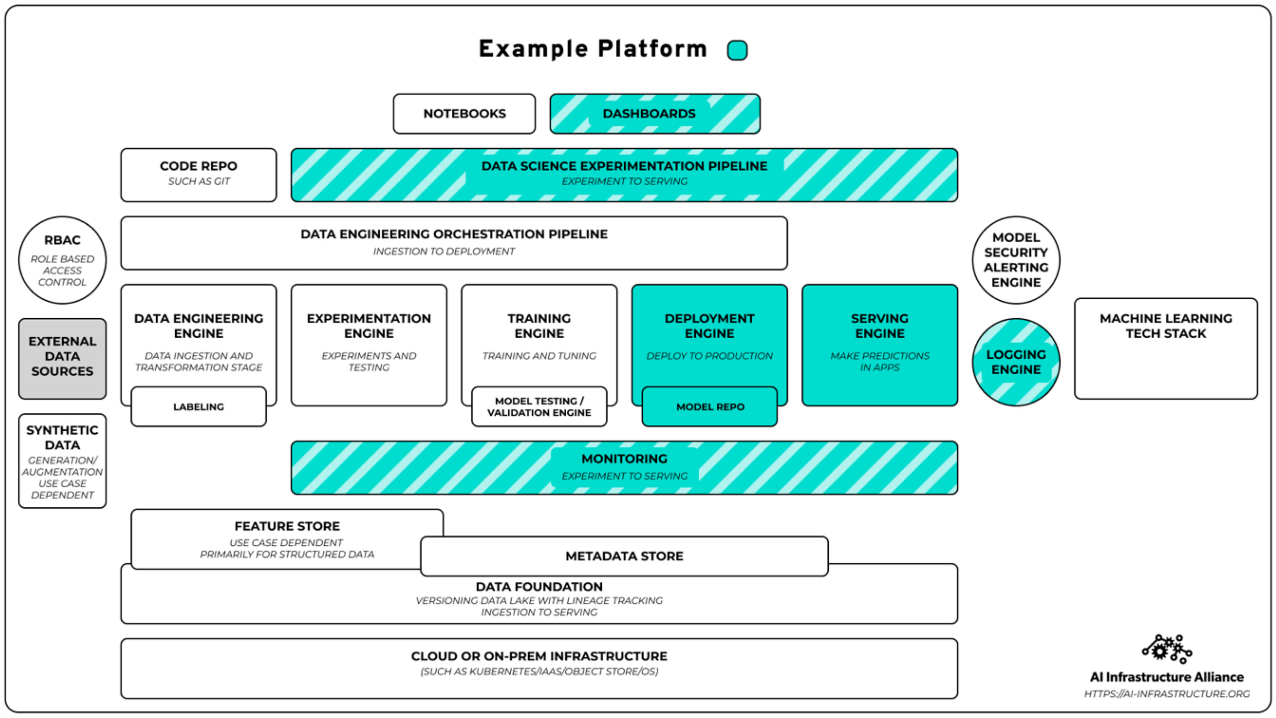
To really help organizations understand AI infrastructure, we've come up with clear category abstractions that do not have many edge cases or overlap. For instance, computer vision is a sub-type of machine learning, but it doesn't have much to do with the capabilities of AI-infrastructure software. Many different platforms listed here support the building and training of computer-vision models along with other kinds of models. "Labeling platforms" is a baseline category that describes an entire range of capabilities in AI-infrastructure software.

We've taken a two-fold approach to helping companies to clearly understand the capabilities of the platforms in the ecosystem.

First, we've distilled the capabilities down to the point where it would be difficult to boil them down any further, and we added them to a blueprint that represents an idealized stack. Vendors do not fit neatly into any single box but may have capabilities across multiple boxes.


NOTEBOOKS
DASHBOARDS
CODE REPO
SUCH AS GIT
DATA SCIENCE EXPERIMENTATION PIPELINE
EXPERIMENT TO SERVING
RBAC
ROLE BASED ACCESS CONTROL
DATA ENGINEERING ORCHESTRATION PIPELINE
INGESTION TO DEPLOYMENT
MODEL SECURITY ALERTING ENGINE
EXTERNAL DATA SOURCES
DATA ENGINEERING ENGINE
DATA INGESTION AND TRANSFORMATION STAGE
LABELING
EXPERIMENTATION ENGINE
EXPERIMENTS AND TESTING
TRAINING ENGINE
TRAINING AND TUNING
MODEL TESTING / VALIDATION ENGINE
DEPLOYMENT ENGINE
DEPLOY TO PRODUCTION
MODEL REPO
SERVING ENGINE
MAKE PREDICTIONS IN APPS
LOGGING ENGINE
MACHINE LEARNING TECH STACK
SYNTHETIC DATA
GENERATION/ AUGMENTATION USE CASE DEPENDENT
MONITORING
EXPERIMENT TO SERVING
FEATURE STORE
USE CASE DEPENDENT PRIMARILY FOR STRUCTURED DATA
METADATA STORE
DATA FOUNDATION
VERSIONING DATA LAKE WITH LINEAGE TRACKING INGESTION TO SERVING
CLOUD OR ON-PREM INFRASTRUCTURE
(SUCH AS KUBERNETES/IAAS/OBJECT STORE/OS)
AI Infrastructure Alliance
HTTPS://AI-INFRASTRUCTURE.ORG

Even a platform that functions first and foremost as a model serving framework has capabilities in other parts of the blueprint. Solidly colored boxes indicate complete support and focus for that set of capabilities, while striped shading indicates a partial set of capabilities in that area.


Example Platform
NOTEBOOKS
DASHBOARDS
CODE REPO
SUCH AS GIT
DATA SCIENCE EXPERIMENTATION PIPELINE
EXPERIMENT TO SERVING
RBAC
ROLE BASED ACCESS CONTROL
DATA ENGINEERING ORCHESTRATION PIPELINE
INGESTION TO DEPLOYMENT
MODEL SECURITY ALERTING ENGINE
EXTERNAL DATA SOURCES
DATA ENGINEERING ENGINE
DATA INGESTION AND TRANSFORMATION STAGE
LABELING
EXPERIMENTATION ENGINE
EXPERIMENTS AND TESTING
TRAINING ENGINE
TRAINING AND TUNING
MODEL TESTING / VALIDATION ENGINE
DEPLOYMENT ENGINE
DEPLOY TO PRODUCTION
MODEL REPO
SERVING ENGINE
MAKE PREDICTIONS IN APPS
LOGGING ENGINE
MACHINE LEARNING TECH STACK
SYNTHETIC DATA
GENERATION/ AUGMENTATION USE CASE DEPENDENT
MONITORING
EXPERIMENT TO SERVING
FEATURE STORE
USE CASE DEPENDENT PRIMARILY FOR STRUCTURED DATA
METADATA STORE
DATA FOUNDATION
VERSIONING DATA LAKE WITH LINEAGE TRACKING INGESTION TO SERVING
CLOUD OR ON-PREM INFRASTRUCTURE
(SUCH AS KUBERNETES/IAAS/OBJECT STORE/OS)
AI Infrastructure Alliance
HTTPS://AI-INFRASTRUCTURE.ORG

Second, we created a new kind of landscape guide, unique to the AIIA, that includes a high-level feature matrix, and we include vendor capabilities across each of the categories rather than pigeonholing them as having only one capability. We also rate the vendors on whether they offer complete support for a category or partial support. For instance, a company might have explainability features but not robust monitoring and observability features, so it would only rate as partial support.

With those abstractions in mind, we break down the software contenders in each of the major categories below. We go over the major companies, startups and open-source platforms, as well as the various capabilities to expect in each category. We also explore the development of each of these capabilities. Some are highly developed and some less so. Lastly, we give you a projection of how these capabilities will develop over the next five years and what gaps need to be filled in for missing capabilities.

ORCHESTRATION, PIPELINES, COMPUTE ENGINES

Companies and platforms covered in this section include:

Apache Spark, Ray, Kubernetes, Databricks, Argo, Airflow, HPE, ClearML, Pachyderm, Comet ML, Neu.ro, DAGsHub, Google Vertex, Amazon SageMaker, Azure Machine Learning, Valohai, Arrikto, Modzy, Kubeflow, Iguazio, Neptune AI, Infuse AI, Dbt, Flyte, Domino Data Labs, Dataiku, Prefect, Weights & Biases, H2O, ZenML

Orchestrators are the control engines in the AI/ML development lifecycle. They organize everything from how data is ingested, cleaned and transformed, to the training and tuning models, to the deployment of finished models. In short, they're the puppet masters of your AI/ML workflow and one of the contenders for your core platform choice.

The terms **orchestrator** and pipelines are often used interchangeably, and though they are similar terms, they have subtle differences. Orchestration involves the command and control of the various steps, whereas a pipeline is the series of steps themselves. It's worth noting that while pipeline has become the standard term in the industry, it's a bit of a misnomer. The movement of data and models from ingestion to deployment is often a DAG or a decision tree with lots of branching steps, and it is also cyclical in that a model is never really done, but rather moves back through the process to learn from new training data or to get tuned and updated.

Nevertheless, we've chosen to stick with the standard terminology instead of inventing new terms so as not to create confusion. We've also chosen to use the words orchestration and pipelines together to indicate both the flow of the data, code and models through their journey from beginning to end and back again, as well as the modules that control that flow. Unlike with traditional coding pipelines, most notably continuous integration/continuous delivery (CI/CD) pipelines, which only track code and automated tests, AI/ML pipelines track code, tests, training and the movement and transformation of data.

Returning to the blueprint for a moment, you may have noticed that the AIIA divides orchestration pipelines into two areas of primary focus:

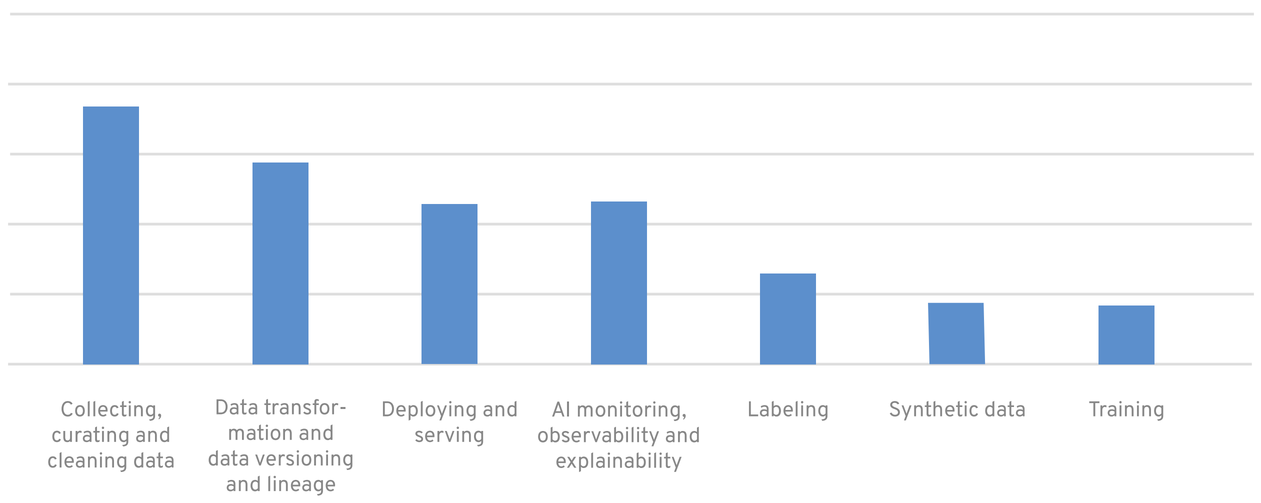
- Experimentation Pipelines
- Data Engineering Pipelines

Experimentation pipelines are heavily focused on data science workflows. In an experimentation pipeline, a data scientist runs different experiments, trains various versions of models, often in parallel, and packages up models for production to a serving engine. The primary focus is experimentation and building models. Notice that they tend to run from data cleaning, after data has been ingested and transformed, all the way through training and deployment.

Data engineering pipelines are more focused on the data engineer persona. Data engineers ingest data from various data sources, clean it, transform it, check it for errors and prepare it for use by data scientists, who tend to focus on data at a higher level of abstraction. Of course, there are data scientists who act as data engineers as well, but in more advanced teams we find the roles increasingly specialized. Data pipelines run from ingestion to packaging models up for production. At the end, they tend to overlap with experimentation pipelines in deployment, but they tend to do the underlying system's work of deployment, such as packaging the model up with dependencies and scheduling it to a container or serving engine.

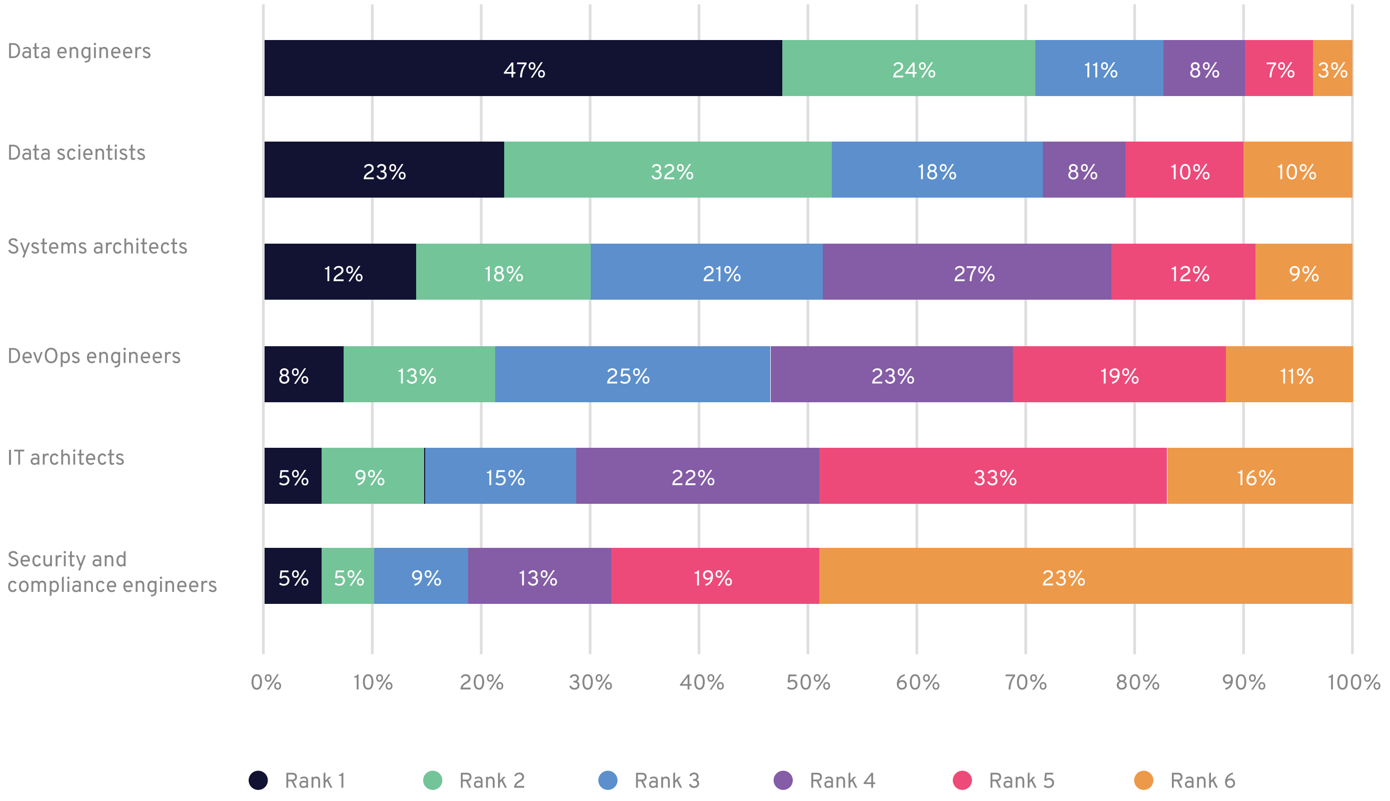
We've discovered that the teams surveyed often faced their biggest challenges with collecting and cleaning data, QAing and transforming data, which falls squarely on the shoulders of data engineers.

Where have you faced the biggest challenges in productionizing models?



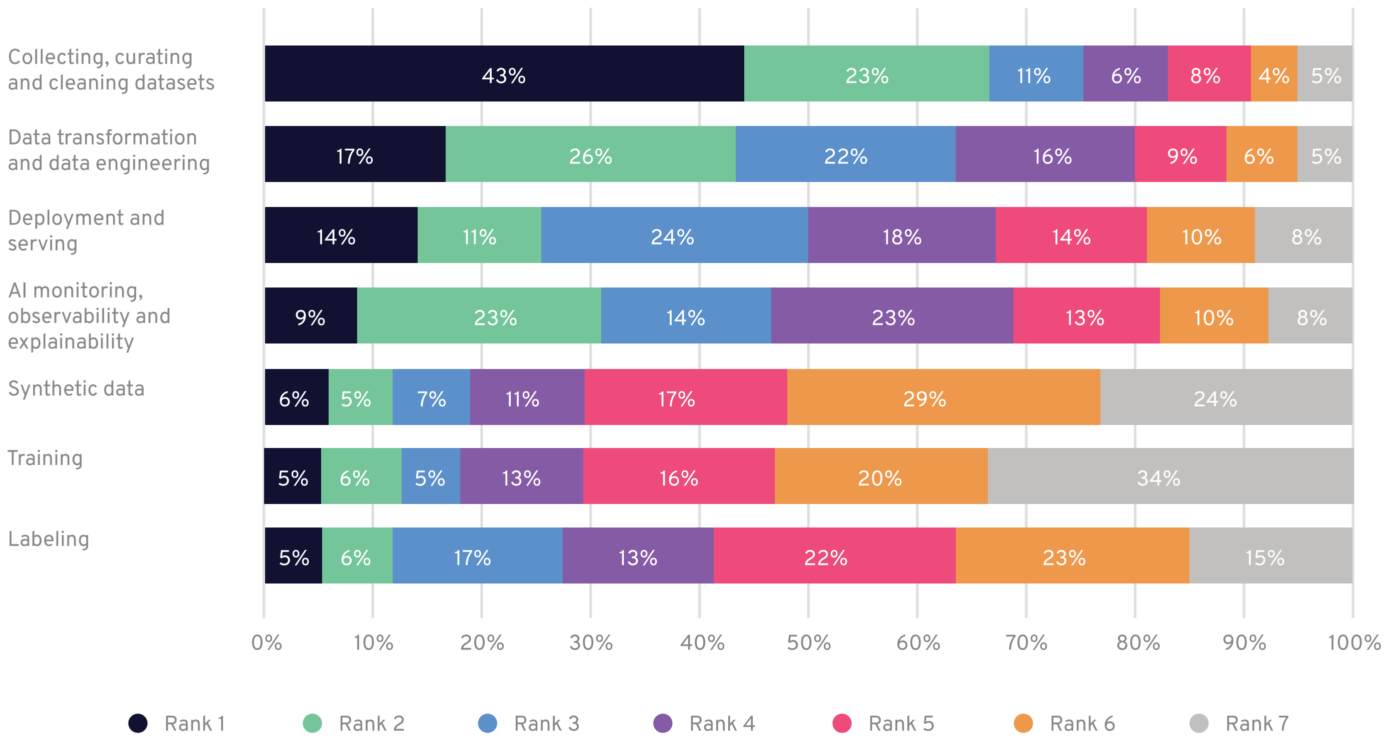
This is reflected in the composition of teams as well. Most companies surveyed employed more data engineers than data scientists.

What is the composition of your AI/ML team? Rank from most employees or consultants to fewest employees or consultants. (Ranked first)



It's also reflected in where teams are spending the most time and money.

What parts of the AI/ML infrastructure should receive the most resources (i.e., talent, time, money)? Rank in order from most to least. (Ranked first)



To further illustrate the difference between the two roles, consider the following: A data engineer might be more worried about connecting to external data sources via RBAC tokens, ingesting that data from Snowflake or Amazon Redshift or an object store like Amazon S3 and transforming the data into a format that is readable by the tools a data scientist is using, such as Pytorch or Tensorflow. On the other hand, a data scientist is largely concerned with data at a higher level. A data scientist focuses on understanding the features of the data, such as calculating age from date of birth or discovering the clusters of pixels that indicate the outlines of a person or a building in an image.

This is roughly equivalent to the old systems administrator and programmer dichotomy in traditional programming, though it is not a perfect analogy. Today's data engineers have the programming skills necessary to automate many aspects of their work, and many data scientists are comfortable altering data to make it ready for feature extraction.

In practice, there is often a lot of overlap between these two styles of pipelines and many platforms do aspects of both data engineering and experimentation but it is helpful to consider them separately because in our experience platforms tend to lean towards one or the other end of the spectrum and that affects how their interface, API, programming and visualizations manifest.

The Two Types of Orchestration

There are two major types of orchestration:

- Loosely coupled
- Tightly coupled

Both have their advantages and disadvantages. The basic difference between them is simple. Loosely coupled orchestration engines are not tied to the underlying execution. Airflow is an example of a loosely coupled orchestrator. It does not have a dedicated compute and scheduling engine. Because it is loosely coupled, it can execute tasks on a diverse set of compute engines, such as Kubernetes or Spark, and it can even execute other distributed application frameworks like Ray or Iguazio's MLRun.

Tightly coupled orchestration systems are tightly bound to their underlying compute engine. Spark is a perfect example, as it is both an orchestrator and a distributed big-data processing engine. Pachyderm is another example of an orchestrator that is tightly coupled with its underlying execution engine.

The biggest advantage of a loosely coupled framework is that it is general purpose and can execute tasks on many different engines. It can also execute the tasks of other tightly coupled orchestrators. That makes it useful as an orchestrator of orchestrators. Loosely coupled frameworks tend to be higher level and more abstracted in terms of the kinds of actions they can perform. An example of a loosely coupled framework is Airflow.

The biggest disadvantage of a loosely coupled framework is that because it is more general purpose, it can't take advantage of all the unique underlying execution engine capabilities without plugins. Even with plugins, a loosely coupled orchestrator is not a two-way street. Because Spark has the knowledge of the underlying data, it can share data between tasks. It also has knowledge of memory allocation and compute resources, and different scheduling options that it can't share with Airflow.

Loosely coupled orchestrators often have no concept of the underlying data and so can't trigger pipelines based on changes in data. On the other hand, tightly coupled systems, such as ClearML, can trigger pipelines based on changes to the data, such as when new telematics data flows into a data lake, which in turn triggers a job with only changed data instead of the entire dataset. There are exceptions to this rule, such as the loosely coupled orchestrator, Prefect, which can pass state and data dependencies to underlying systems, but loosely coupled systems are always at a disadvantage to tightly coupled systems when they need to utilize a special feature of the underlying system.

The disadvantage of tightly coupled systems is that their orchestration often cannot extend to other systems to execute tasks, and you must deploy the underlying compute engine to make them work. Their orchestration is siloed to that system.

Which style of orchestration system to use depends on an organization's needs and scale. Smaller teams may only need the capabilities of a specific engine and thus focus all their energy on that engine to meet their needs. Large enterprise

AI/ML teams, with diverse model production needs, are likely to need more than one compute engine for different kinds of AI/ML capabilities, and so they may use a combination of loosely coupled frameworks, as well as the underlying tightly coupled framework at different stages in the machine-learning pipeline.

Compute Engines

Now that we've explored orchestration, let's turn to the underlying compute engines. The bedrock of any AI/ML workload is its compute engine. These are the workhorses of execution in any AI/ML platform. They process data and code; schedule resources like memory, disk space, GPU, TPU and CPU; execute steps; and scale and parallelize execution of those steps.

We're going to assume you have a generalized understanding of virtualization, containers and parallelization in general and not go deeply into the details of these systems. But they are important to reflect on for a moment because we've found that the vast majority of current AI/ML platforms rely on one of three compute engines:

- Kubernetes
- Spark
- Ray

There are others, but it is worth focusing on these three as they encapsulate the three key focuses a compute engine can have, namely:

- General purpose compute engine
- Tightly coupled orchestration and compute
- Machine-learning application focused compute

There are some platforms that run "bare metal" in theory, but today that mostly means running on virtualized containers or serverless instances in the cloud or on virtualized or serverless local instances in a data center.

Of the three, Kubernetes is the most general purpose compute engine. It evolved from Google's own internal container and orchestration capabilities and the desire to build a clean, universal cloud "operating system." You can see its history and timeline here. Kubernetes is generic enough that it can encapsulate and run other compute engines on top of it, making it the most general purpose of the compute engines. It is language agnostic, allowing teams to run any kind of code, including Bash, C/C++, Python, Rust and Java. It also does not matter if you are building a web-scale distributed web application or a machine-learning training pipeline, it can run any kind of workload because it is so general purpose.

Apache Spark is the second most well-known compute engine, and it has a long history. Spark has its own battle-tested parallel processing engine and does not rely on third-party scaling platforms like Kubernetes, although Spark can be deployed on Kubernetes. It's important to note that Spark was created before the AI/ML deep-learning revolution that started with AlexNet. It was primarily designed to deal with processing big data, based on Google's MapReduce paper and as a better version of Hadoop, an early disk-bound version of a MapReduce-type system. Spark's biggest innovation was keeping and processing most of the data in memory, which made it up to one hundred times faster than Hadoop, which was heavily disk dependent. It's a testament to the design of the system that it allows for additional use cases like AI/ML, but it is not always a perfect fit.

Spark is most adept at dealing with structured data, such as columnar text data in a database, and semi-structured data, like JSON files, though there have been efforts to make it more capable at dealing with unstructured data, like video, audio, images and unstructured text like legal documents, specifically through the Delta Lake project, though that is still not the primary strength of the platform.

As noted earlier, Spark is both an orchestrator/pipelining system and a compute engine. Its orchestration is built around Scala, though it does support other languages through ports and wrappers, such as Python.

Ray is the third compute framework and a relative newcomer to the market. It was designed at UC Berkeley's RISELab, primarily as a tool for reinforcement learning (RL) and as a "replacement for Spark" with a specific ML focus, since many kinds of ML jobs do not fit into MapReduce style paradigms. As computer science professor Michael Jordon wrote in an article from 2017, in the early days of the project:

> You need flexibility. You need to…put together not just things like neural nets but planning and search and simulation. This creates all kinds of complex task dependencies. It's not very easy simply to write a MapReduce kind of paradigm. You can write it, but it's not going to execute very effectively when you have very heterogeneous workloads and tasks. It needs to be adapted to the performance of the algorithms, as the system is learning, to change its flow, its tasks.

The platform has since been adapted to be a more general purpose compute framework for machine learning through Dataset, a distributed data loading and compute library, though it is still most often used for reinforcement learning and as a serving engine.

Ray is perhaps the newest and most cutting-edge of the frameworks, but it is simultaneously the least general purpose and the most specifically designed for ML workloads. It should be noted that calling Ray a replacement for Spark is not exactly correct. Ray does not have its own underlying compute engine, and it is not low-level enough to act as one currently. It relies on general purpose compute frameworks like Kubernetes to run. Instead, it is a set of Python libraries for building distributed machine-learning applications and serving them. It focuses on some of the key dependencies for newer and more cutting-edge AI/ML techniques that do not fit into the do-a-task-and-then-wait paradigm of MapReduce. For example, reinforcement learning often has many parallel and less linear tasks that work together in a dependency tree, and they all need to finish before the next step can proceed. Lastly, it is exclusively for Python applications.

Another potential advantage of Ray is you can deploy a single node without needing to deploy Kubernetes. This gives developers the chance to write code, test it and then scale it out to Kubernetes clusters without changing that code later, which lowers the barrier to entry. However, according to the documentation, "Ray Serve lacks the ability to declaratively configure your ML application via YAML files. In Ray Serve, you configure everything by Python code."

As noted earlier, we can think of the three kinds of platforms in different ways. Kubernetes is the most low-level. It is totally general purpose, and it does not matter if you are building a web application on top of it or a machine-learning training pipeline, but because of that, the logic of those applications exists higher up, so it is not enough by itself to create a machine-learning orchestrator and pipeline. Spark is the most long-lived and mature of the tightly coupled orchestrators, but it was not designed with AI/ML in mind. It has built in scaling and processing capabilities, and at its best it can process data that fits cleanly in a database with great flexibility and power. Ray is the newest of the frameworks and was designed with AI/ML in mind, and it takes into account the latest techniques in the space, such as RL, but it is not an execution engine in its own right and needs something like Kubernetes to run on.

Understanding the capabilities of those engines helps us understand the offerings of many platforms on the market because they sit at the base of those platforms and inform their capabilities higher up the stack. Understanding the limits and peculiarities of those platforms in particular helps us see through some of the marketing claims of various offerings, especially when marketing teams promise capabilities that are not an easy fit within the limitations of each style of platform.

Data Engineering Orchestration and Pipelines

Kubeflow is one of the earliest and most well-known orchestration/pipelining systems. It gestated at Google as an open-source project designed to work on Kubernetes, which is the de facto standard for cloud application hosting and scaling. Kubeflow is not a single project but a collection of projects. When referring to Kubeflow here, we are referring only to Kubeflow pipelines, which is by far the most popular project in the group, along with notebooks. Kubeflow pipelines themselves are largely based on Argo Workflows.

In relation to the AIIA blueprint, Kubeflow most closely fits the data engineering pipeline definition and that is reflected in the usage of the project, roughly 73% of which is by ML engineers, another term for data engineers, as of 2021. The Kubeflow project predates the concept of MLOps, and it reflects a more DevOps workflow in its design. It does not include a concept of a data-driven pipeline, where new datums are able to trigger events and kick off pipelines and automation steps. It also lacks pure language agnosticism, with a heavy focus on Python. R support is rudimentary. It also lacks commercial support, which means any team running it in production will need to support it completely on their own or through the community.

Several commercial products have built their stack to include Kubeflow, most notably Google's Vertex AI, HPE's Greenlake for MLOps and Arrikto's MLOps platform. As Kubeflow pipelines are still in active development and often tricky to support, the AIIA does not recommend deploying the open-source version in production at this time, unless you have a strong open-source team with a history of supporting open-source projects for mission-critical applications. To date, no company has specifically created a dedicated, supported, standalone commercial version of pipelines. Instead, pipelines are wrapped into a larger tooling structure, such as Arrikto's data snapshotting filesystem, Google's AutoML tools and HPE's suite of open-source tools for AI/ML in Greenlake. For teams that are not highly sophisticated in supporting upstream open-source projects, we recommend choosing a vendor that can support Kubeflow as part of a package with rapid bug fixes, a clean upgrade path and regression patching of earlier versions.

AirFlow is another data-engineering-focused pipelining system and allows users to build DAG objects in Python to define their workflow as code. As mentioned earlier, it is loosely coupled. Many companies and organizations use Airflow for CI/CD-style orchestration of AI/ML pipelines, but it wasn't purpose built for AI. It is 100% Python focused and doesn't allow any easy way to plug in other languages. Airflow works best with workflows that are mostly static and slowly changing. It is not built for large quantities of data from one task to the next, and the project is not recommended for "high-volume, data-intensive tasks," according to the project readme. Furthermore, "a best practice is to delegate to external services specializing in that type of work." Also, "Airflow is not a streaming solution, but it is often used to process real-time data, pulling data off streams in batches."

Prefect was developed specifically to address some of Airflow's limitations. Most notably, it's a loosely coupled framework that takes the underlying data dependencies into account as well as the task state, whereas Airflow accounts only for state. It also adds capabilities like task versioning. It is a Python library that looks to minimize extra dependencies like its own scheduler. To run production or parallelizable workflows, it relies on Dask to run distributed.

Dbt is another data-engineering-focused transformation engine, but it is not a pipelining engine, allowing for a series of orchestrated steps. It is exclusively focused on SQL and running those queries directly in data warehouses, so it is mostly useful for BI and analytics but is not an ideal choice for AI/ML workloads; it has no support for unstructured workloads and should not be considered when looking for general orchestrators, either loosely coupled or tightly coupled, but it can be very useful for streamlining working with SQL backends.

Pachyderm is a tightly coupled data-engineering-focused orchestrator/pipeline that also combines versioning and lineage tracking with an immutable data lake that includes data deduplication. It allows users to string together complex transformation steps in any language because it is container-based and thus language agnostic. It relies on Kubernetes execution to scale, unlike Airflow, which can run with or without Kubernetes. Pachyderm excels at unstructured data like video, audio and imagery, and semi-structured data like JSON, but it can handle structured data as well, such as CSV files. While there is an advantage to keeping all data in a unified data lake, organizations that are heavily reliant on structured data will find databases more performant for highly transactional processing. Pachyderm automatically builds a DAG, rather than requiring the user to pre-build it, by treating each step in the pipeline as an atomic unit of work by using well-defined JSON or YAML definitions to define steps in its pipelines. Those definitions call individual containers to execute code to transform, train and track models as they move through the ML lifecycle. However, while Pachyderm can do training and deploy models, it is primarily used in complex data preprocessing and data preparation. A team with complex training requirements should turn to frameworks like Horovod or Determined AI (now part of HPE). When it comes to deployment, a dedicated deployment framework such as Algorithmia (now part of DataRobot), Ubiops, ML-Run or Seldon is usually used.

As noted earlier, Apache Spark is both a processing/compute engine and a data engineering pipeline, though the tool can also be used for data science experimentation. We'll focus on its pipelining capabilities here. It's important to differentiate between Spark, an open-source tool with multiple vendors such as Microsoft offering it as a service, and Databricks, which is the primary corporate backer of the project and the largest player in Spark deployments. Spark also underpins many other platforms as the scaling and processing engine, as in some parts of the DataRobot AI cloud. It's important to understand that the capabilities of these various platforms are highly dependent on what they built on top of Spark, namely their various APIs and GUIs as well as their proprietary overlay applications. When you're evaluating Spark as a pipelining tool versus a company backing Spark, you are evaluating very different things. In many ways, you should see Spark as a choice of compute engine with some core capabilities and limitations and then evaluate the vendor platforms built on top of Spark as a separate consideration.

Let's start with Databricks's version of Spark. While Databricks's offerings have morphed into a suite of products that supports everything from ML to analytics to visualizations, Spark was primarily designed as a replacement for Hadoop, with a focus on data engineers working with big data. While Databricks shifted much of its focus to AI in its marketing, the primary use case for Spark was for big data and analytics workloads for most of its history. Because it uses Parquet files for storing data, which are columnar database files, it excels at structured and semi-structured workloads. While Databricks does say it's possible to use Spark as a unified storage backend for any kind of data with its lakehouse architecture, in practice this is rarely done if an organization is dealing with large audio, video or image files (such as high-resolution Satellite images), as storing unstructured data in Parquet files is largely impractical, even with the compression capabilities of the format. However, Databricks's Spark is one of the most popular platforms for structured and semi-structured batch workloads, and it has a large community supporting it. In addition, Databricks's Spark and their suite of tools surrounding it makes them one of the most well-supported and well-funded companies in the space for high-speed structured data processing and structured AI/ML workloads, such as churn prediction, demand forecasting and anomaly detection.

Scala is the primary language underpinning Spark, and though Databricks supports Python, which has emerged as the undisputed lingua franca of ML, the support of key Python libraries requires porting by Databricks which largely act as wrappers around Scala. This can create some anomalies in error debugging, and it means that teams may need to wait for a specific Python library or specific features of a Python library to be ported, rather than simply getting the latest Python library from its source repo and interacting with a system natively. That said, Databricks's Python libraries are one of their most consistently updated and widely supported.

Now let's turn to DataRobot, one of the few systems that looks to focus on both data engineering pipelines and experimentation pipelines. Again, we shouldn't think of DataRobot as a single product but as a suite of tools. Some of these tools use different pipelining backends. The data-engineering-focused tool in the AI cloud is called Data Mesh and came out of an acquisition of Paxata. The tool has a strong GUI, and it excels at data prep for ML, though it is not a generic data engineering pipeline for any kind of transformation. It uses a set of data connectors to a variety of backends, such as Amazon Redshift, Snowflake, MySQL, Oracle and SAP HANA. Since most of the data connectors are for databases, DataRobot is largely focused on structured and semi-structured data and most of their excellent visualization tools support columnar data, but they can work with unstructured data as well. We'll discuss the data science aspects of the DataRobot platform in the next section on experimentation pipelines. While DataRobot existed as a fully integrated set of tools and as a tightly coupled pipelining system, the team worked to decouple the tooling over the last few years so that outside tools can be incorporated into the workflow more easily.

Amazon's Data Wrangler is part of the SageMaker toolset. Like DataRobot, SageMaker is not a single unified tool but a set of tools. Data Wrangler itself is less a generic pipelining tool and more of a visual way to do data transformations. It focuses completely on data cleaning, transformation and prep and on nearly 100% structured data transformations. It includes a wide variety of connectors and over 300 pre-baked transforms, such as one-hot encodes, and gives users the ability to design their own data wrangling steps as well. It is not a generic data transformer, and it is strictly focused on creating as many pre-built data science transforms as possible for structured data. The largest challenge with using SageMaker is that it is designed as a totally standalone suite, unlike other tools that strive for easy connectivity to third-party tools, and thus it is difficult to use it with third-party external tools. However, if a third-party provider has either a built-in integration or provides an API, then it is possible to use them with SageMaker.

Azure Machine Learning is the most open-source focused of the big cloud providers. It offers two distinct sets of proprietary pipelines and an open-source offering. Its Data Factory pipelines are focused on data ingestion and transformation, and they have many pre-built connectors for everything from major databases like Oracle, Snowflake and MS SQL server but also for files via FTP and NFS and object stores. Their Data Factory also includes predefined transformation steps that can ease the process of ingesting and transforming data. Their open-source alternative to Data Factory is Airflow.

Dataiku has some of the most advanced visual tooling for constructing data pipelines in a GUI, and it is able to act as a transformer for many kinds of data, though, again, the primary focus is on structured data. Its GUIs provide excellent dashboards and visualizations along with advanced markdown capabilities for columnar data and text. It does provide a series of plugins to deal with unstructured data, but they are largely either Tier 2 supported or unsupported, which means they are used at an organization's own risk. Dataiku does provide some examples of unstructured use cases for deep learning in its knowledge base, but they are mostly pre-baked solutions such as object detection with a pretrained model. Dataiku uses a mixture of pipelines on the backend to accomplish its goals. Under the hood, it can use its own pipeline engine, Hadoop/Spark or Kubernetes/Docker, and it can run computation directly in SQL databases such as Oracle or Snowflake. The platform provides one of the largest arrays of connectors for external data, from widely supported platforms common to most data transformation engines, such as Amazon Redshift, to less common ones like those pulling data directly from Twitter.

The last data-engineering-focused pipeline worth a minor mention is Dagger, from the former creators of Docker. It includes some novel features, most notably the idea of portable container pipelines. Similar to Pachyderm, it uses templating to define steps in the pipeline, which also makes it language agnostic. It uses Google's CUE configuration language framework instead of YAML or JSON. Dagger does not include any kind of data storage or any concept of data-driven pipelines. Lastly, Dagger was only launched recently, and it should be considered only for early adopters at this time.

Experimentation Pipelines

By far the largest design pattern in AI/ML platforms is experimentation pipelines that focus on data scientists. These pipelines are usually DAGs (directed acyclic graphs), which are basically a conceptual representation of a series of steps or tasks, representing a mathematical abstraction of a data pipeline. Experimental pipelines tend to hide the actual DAG (a creation by the user) and include a well-known interface, such as Jupyter Notebooks, as the primary way to interact with the system. They do often allow the creation of steps through Python, the default language for most AI/ML tasks. By contrast, data-engineering-focused pipelines often require DAGs to be written directly.

Almost every AI/ML platform includes some notion of an experimentation pipeline, which tends to blur the lines between engineering-focused pipelines and experimentation pipelines. For instance, despite being used mostly by data engineers, Kubeflow includes support for Jupyter Notebooks, as does Pachyderm and Arrikto. Nevertheless, we maintain the distinction at the AIIA by asking the question, where does the primary user of the platform spend their time?

If the primary user is creating containers, writing steps to the pipeline in YAML or JSON and creating DAGs, then they're mostly likely data engineers, and we would classify the purpose of the pipeline as data engineering. If most users are working in notebooks, running experiments, tuning hyperparameters and building models, then it's mostly data scientists using the platform, and we would classify it as an experimentation pipeline. Just be aware that many of the pipelines can function in both arenas, and you will have to make the choice based on the skills and composition of your team.

As a side note, while many of the platforms discussed in this report use the phrase end-to-end in their marketing, the AIIA recommends that organizations adopt a more comprehensive definition that includes the work of data engineers and data scientists as a whole and accept that a collection of tools may be needed to serve both groups well. As such, organizations should evaluate the capabilities of any platform along those lines to decide if it truly has end-to-end capabilities.

As such, most experimentation pipelines largely assume that the data is prepped and largely ready to go. They include data ingestion capabilities or the ability to point to stored data but generally do not include the ability to do steps like imputing missing or corrupted data, transcoding audio from WAV to MP4, resizing all images or changing images to a different format, which is the work of data engineers. Again, sometimes they include this ability, but in general, advanced concepts of ETL are not part of experimentation-focused platforms. Instead, they focus on helping data scientists extract features, testing multiple models against each other, tuning hyperparameters and finalizing a model for production.

Experimentation engines tend to fall into two categories:

- Platforms that include their own orchestration/pipelining engines and/or schedulers
- Platforms that aggregate metadata from other pipelining systems and orchestrate and/or visualize those pipelines

ClearML, DataRobot, Google Vertex, Amazon SageMaker, Azure Machine Learning, Neu.ro, Valohai, Iguazio, Flyte and

ZenML fall into the first category, in that they have their own pipelining engines. While Neu.ro includes its own pipelining engine, it primarily sees itself as an orchestrator and integrator of third-party tooling, so we place it here. Companies that focus on third-party integrations and look to act as a glue layer could constitute a third category, however since many platforms focus on integrations with third parties, such as Dataiku, Iguazio and Domino Data Labs, we decided it would not add much value.

Weights & Biases, Neptune AI, Comet.ML, Infuse AI's Piperider, Domino Data Labs, Infuse AI and DAGsHub would broadly fall into the third category in that they rely on other pipelining systems, such as Spark, to run. They put their emphasis on visualizations and tracking experiments across different tools.

You can usually recognize the second style of platform by a telltale sign in their marketing that usually reads something along the lines of "integration with a single line of code." Metadata stores are designed to be easily integrated with other platforms, and they look to act as a single source of truth across the disparate metadata of the systems they connect with.

Let's look at the platforms in turn, starting with the loosely and tightly coupled orchestrators.

Loosely Coupled and Tightly Coupled Orchestrators

ClearML is an open-source suite of tools that cover a broad range of AI/ML tasks. Its orchestration and pipelines serve both data engineering and experimentation, with an eye towards making the transition between the two simple and easy. The orchestration module introduces scheduling and remote job execution on bare-metal machines, cloud infrastructure and Kubernetes clusters. The UI is geared towards building and testing models and getting them into production, allowing job scheduling directly from the experimentation UI. It also includes tools like a metadata index for data that allows data scientists to search through that data as well as split and sort it. The tools allow for classic data science tasks like slicing up training data and testing data on the fly, along with cloning previous experiments. Along the way, the platform develops artifacts from code and versions of all models created into a queryable model repository. It allows integrations with Tensorboard, Matplotlib and Git. Lastly, it allows for hyperparameter optimization on multiple machines, data preprocessing and model deployment, giving data scientists a more end-to-end experience to get models into production.

DataRobot's experimentation capabilities are largely built around Spark, but they also have a number of excellent proprietary interfaces that make the data science experience easier, particularly for structured use cases and extensive visualizations. In addition, DataRobot has one of the better AutoML capabilities on the market. We do not spend much time discussing AutoML in this report because it calls up ideas of AI that can automatically do most of the work of a data scientist, when that is not the reality. However, DataRobot's vision of AutoML is quite focused and clear. It essentially involves trying out a lot of known solutions to a specific problem in ML all at once and offering users the ability to see which one works best without a lot of manual experimentation. A data science team can then focus on refining or fine tuning various methods from there. Like other large platforms in the space, DataRobot wants to create a true end-to-end experience, and their tools not only deal with experimentation but also with some aspects of data engineering, as well as deployment, serving and monitoring.

Vertex AI is Google's competitor to SageMaker and the Azure machine-learning cloud platform. Like other big platforms, it's really a suite of loosely connected tools rather than a seamless and unified experience. It's one of the only commercial platforms built on top of Kubeflow pipelines (and Tensorflow Extended), but it also uses a lot of proprietary tech under the hood, such as neural architecture search and their own feature store. Unlike some of the other platforms it competes with, it doesn't focus as heavily on a tremendous GUI design, and that's never been Google's strongest asset anyway. But it does leverage Google's know how with containers, large scale systems and the command line. It's somewhat more modular than SageMaker, but it is still largely a walled garden, though its API is stronger than some more monolithic engines. The system is focused on teams with heavy programming and command-line experience.

H2O is an open-source, in-memory big-data processing engine for machine-learning and analytics workloads that competes with Spark. Like many other large platforms profiled in this report, it's best to think of it as a suite of tools with the in-memory pipeline as the core behind it. It's best suited to structured data workloads, but it does have support for images, audio and other unstructured data, and it can read common third-party formats like Parquet files. It supports many common ML algorithms, with the developers custom writing them for distributed processing. That means that H2O is mostly used for running predefined machine-learning models versus developing your own neural networks. H2O's framework predates Pytorch and other popular frameworks, and it competes with more widely adopted frameworks like Pytorch ML, in addition to being a fast pipelining system. It consists of four primary products: H2O, its proprietary in-memory pipeline, Deep Water, which integrates it and allows it to leverage Tensorflow, and Sparkling Water, which integrates with Apache Spark. Driverless AI is an AutoML tool that competes with DataRobot's offering and allows for automatic feature engineering, model selection and tuning.

Amazon SageMaker is a suite of tools that includes everything from data wrangling to pipelines to a proprietary feature store to a labeling system. Its experimentation pipeline runs on the proprietary SageMaker Pipelines and is accessed primarily through SageMaker Studio. The suite of tools inside SageMaker expands regularly. Most of the visual interfaces are designed almost entirely for structured data workloads, and it is not ideally suited to running unstructured workloads. It's currently one of the most walled gardens of all the platforms and is designed to be an all-in-one integrated suite.

Azure Machine Learning splits their experimentation engines into a proprietary pipelining system, Azure Machine Learning pipelines, and a CI/CD-style orchestrator called Azure pipelines, which is comparable to Jenkins. Its pipelines are modular and allow independent execution running on Docker containers. Its orchestration allows the scripting of steps and orchestration of Kubernetes, VMs, Azure Functions, Azure Web Apps and more. The orchestrator uses stages, gates and approvals to create a deployment strategy, and it allows orchestration steps from other CI systems, like Jenkins.

Iguazio's orchestration is built around their open-source MLRun python framework, which executes steps on their Nuclio serverless platform. The orchestrator is loosely coupled and designed to orchestrate a number of platforms, like Spark. It has built-in integrations with Tensorflow, Pytorch and other major machine-learning frameworks. Its interface is weighted towards model experimentation, featuring engineering and training, but it's also excellent at deployment and can serve models as well, and it has built-in integrations with distributed analytics frameworks like Dask.

Neu.ro is a loosely coupled orchestration system that prides itself on being able to easily integrate with other systems and to orchestrate them. Because it was designed with ML in mind, it is more granular than a more high-level and abstract loosely coupled orchestrator like AirFlow, which was designed for many tasks. Because of the development team focus on making it an orchestrator first, it includes many integrations with other tools in the ecosystem like Seldon Alibi, Seldon Core, Prometheus and Grafana, DVC and other platforms, with the goal of knitting them together and weaving them into a single CI/CD cross platform workflow for ML.

HPE's Ezmeral platform leverages a number of open-source platforms, like Spark and MLflow, under the hood, along with Kubernetes for scaling. It also includes Airflow for loosely coupled orchestrations that can execute against Kubeflow pipelines, Spark or other external platforms. Because it leverages open-source platforms so extensively, it is more modular than other platforms and it allows for swapping in components more easily. It includes a proprietary interface called the App Workbench to help connect these different offerings, as well as an API to schedule jobs more easily.

Similar to HPE Ezmeral, Shakudo built their platform on Kubernetes from a suite of open-source tools such as Tensorflow, Spark, MLFlow and Jupyter, while layering a proprietary interface. This is a trend we expect to see continue in the space as open-source stacks mature. Integrating disparate open-source offerings is a challenge, and we expect to see more companies offer a clean interface to these solutions.

Valohai's platform is similar to Pachyderm in that it uses Kubernetes on the backend, YAML/JSON to define steps in the pipeline and containers to execute code. That makes it language agnostic and even framework version agnostic. A customer may use one version of Anaconda in one step and another in a later step, while using a completely different language like Rust in yet another step. The platform's focus is primarily on deep learning, and its UI makes it lean closer to the data science side of the pipeline because it emphasizes experiments and comparing experiments. Because it is language and tool agnostic, it can be used for data engineering workloads, but because it has a well-defined experimentation interface, we classify it primarily as an experimentation pipeline.

Flyte is a tightly coupled framework that came out of Lyft and eventually found a home at the Linux Foundation's LF AI & Data group. It was designed as a replacement for Airflow but with a focus on ML experimentation and deployment, and because it runs on Kubernetes clusters natively, it is considered more tightly coupled. The system looks to avoid YAML configurations for steps, instead focusing on direct code, such as Python code, to orchestrate steps, much like MLRun, though it also supports writing orchestration steps in Java and Scala. Because it is container-based, it can execute steps agnostically, in a manner similar to Pachyderm or Valohai.

ZenML is a loosely coupled Python-based orchestrator that, like Neurolabs, prides itself on a wide range of integrations with everything from Spark to SageMaker to Argo. It was designed as an orchestrator of orchestrators, with the ability to stitch together an MLOps stack from a wide variety of proprietary and open-source components.

Metadata Engines, Experiment Trackers and Visualizers

Now that we've covered the platforms that include their own orchestration engines, we can focus on the platforms that are more metadata focused, for the purpose of acting as systems of record across multiple platforms, and that offer visualizations and tracking.

The first, Weights & Biases, is not an orchestrator or pipeline system itself, but rather it integrates with other orchestrators to offer experiment tracking and visualizations of experiments. It also offers checkpointing and the ability to rerun experiments. Lastly, it monitors CPU and GPU performance.

Comet.ML also emphasizes experimentation and visualization via easy integration with one line of code. It's designed to be simple to integrate into existing pipelines, even home grown ones, and enables teams to track and compare those models more easily. It allows for workspaces that let teams consolidate, manage, collaborate and report on their machine-learning projects and experiments. Comet.ML provides automatic logging for a number of popular Python machine learning frameworks, even if those frameworks don't support logging natively.

Domino Data Labs acts as a metadata store across platforms, and its Workbench platform acts as an easy interface to Spark and Ray compute engines. Its goal is to act as a system of record across platforms and to make interfacing with other platforms easier, so it falls into the loosely coupled visual orchestrator category.

Infuse AI's PipeRider system monitors changes across pipelines and notifies users when they break expectations to help avoid repeated failures. Like the other frameworks in this space, it is designed to be easily integrated across other platforms, starting with DBT and moving to Snowflake, Tensorflow and Weights & Biases and MLflow. Since it is not designed to do the orchestration itself, it can be gradually introduced to a stack.

DAGsHub is the last metadata and visualization platform we'll profile here. It includes the ability to compare and diff files and to create versions of DAGs, and it allows for team collaboration with the commenting, annotating and sharing of DAGs. It offers integrations with third-party tools like Jenkins, and it is designed for modularity.

Current Trends and the Next Five Years

Orchestration and compute are some of the most essential components in an AI/ML. They form the bedrock of how work gets done on the system, and a misstep in choosing the right framework can prove costly, as orchestrators and pipelines connect with all or most other aspects of the system. This is an area in which every company should study the available options extensively and take their time to make sure the platforms match their needs and their future needs.

Many of the companies in this space offer similar capabilities, and companies looking to make a decision on which platform to use should consider the following:

1. The key capabilities of their team
    - Are they strong programmers or do they need a powerful visual GUI or both?
2. The range of capabilities of the platform
3. The kinds of AI/ML projects they need to support
    - Structured, semi-structured, unstructured or all of the above?
4. Connector and integration support
5. Customer team and support
6. The proprietary tooling capabilities
7. Their graphical front end

We recommend that companies carefully consider not just their crrent AI/ML use cases but any future use cases they might want to work with in the future. There are a number of capable orchestration and pipelining platforms on the market, but not every one will meet every need. If you are not careful, you might find that you've selected a powerful system for working with structured data and databases but will struggle with deep learning later. It's likely you may need a suite of tools to accomplish all of your goals as an organization.

It's important to look beyond marketing. Many platforms promise that they can do everything and handle any kind of workload, but be sure to ask companies to demonstrate a range of capabilities and beware of sleight of hand. If a company promises they can handle high-resolution satellite imagery as easily as textual data, be sure to examine examples and case studies that demonstrate that capability in detail.

It is also difficult to know what kinds of capabilities you will need for future workloads, and the limits of platforms may not make themselves known until your team has used the platform for a time and then pivoted to a new use case, only to find out it is challenging to achieve their aims within that platform. It is likely your team may need to use a range of tools, including one or more tightly coupled data engineering frameworks, as well as a loosely coupled orchestrator and an experimentation engine.

The Next Five Years

In the current generation of platforms, Spark is one of the most dominant. That's largely because it is the oldest existing codebase, and companies that built their capabilities around it have had the most time to mature their interfaces, APIs and those capabilities. If you are working with structured workloads, Spark has excellent capabilities and thousands of successful use cases.

However, MapReduce has limitations with regard to AI/ML. Newer AI paradigms like RL have cascading sets of task dependencies that do not easily fit into a MapReduce structure. Unstructured data, despite efforts like Delta Lake, is not an ideal fit on the platform, and that makes it less than ideal for deep learning. Just as Spark was built with data processing in mind, newer frameworks, like Ray or the Google Pathways training architecture, are being built from the ground up with existing AI/ML workloads and more cutting-edge workloads in mind.

As the field of AI/ML advances and new techniques come out of the research labs and become commonplace, we expect the limitations of MapReduce to continue to show their age. We expect more frameworks like Ray and Pathways to emerge over the next five years and for them to compete to become the default distributed processing and distributed AI/ML application framework for the next ten years. We also expect frameworks for AI/ML that are more language agonistic; even though most major work starts in Python, other languages can and do add value to the various steps of the AI/ML workload. We also expect those platforms to absorb newer AI/ML workloads that rely on that underlying logic. That said, we firmly expect Spark to continue to be a powerhouse in structured data applications and to continue to excel at a broad range of other types of BI and analytics.

While we do expect competition on the AI/ML framework front, we do not expect any other general purpose compute engine to emerge and compete with Kubernetes. Kubernetes itself emerged as the de facto winner against other strong contenders like Apache Mesos, Docker Swarm and Rancher. None of them had the broad mindshare and sticking power of Kubernetes or its general purpose capabilities and scaling potential, other than Mesos, which is no longer a serious competitor in the cloud compute space. Kubernetes services are now available on premises and in every major public cloud, and we do not see serious competition to it suddenly emerging. As such, expect Kubernetes to largely act as the underlying compute layer, and AI/ML framework layers to run on top of it or a serverless platform like Nuclio or Amazon's Lambda.

We expect competition at both the data engineering pipeline layer and the experimentation pipeline layer to remain fierce over the next five years. We will likely see some consolidation among the lightweight integration-style platforms like Weights & Biases, and there are a number of excellent visualization platforms already.

Beyond that, we expect some companies to fail or run out of runway due to funding or management or market fit. We also see some acquisitions and mergers narrowing the space. That said, we certainly have not seen the last of new players to the field, as it is an exciting space and still developing, so new players may offset the losses of older players who have failed.

Finally, it's likely that several platforms will begin to pull ahead of the competition, gobbling up market share and customers, but we do not expect the field to be completely decided in the next five years, with one or two totally dominant players sitting at the top of the heap.

MODEL DEPLOYMENT AND SERVING

Companies and platforms covered in this section include:

Seldon, DataRobot, Iguazio, Modzy, ClearML, ONNX, OctoML, UbiOps, Dask, Flask, NVIDIA Triton, Superwise, Cortex, BentoML, Domino Data Labs, LGN, Ray

Introduction

Over the last few decades, traditional code-based application deployments have evolved to allow for faster and more automated rollouts.

We went from large waterfall deployment methodologies, where companies typically deployed new monolithic code once or twice a year, to Agile- and DevOps-style deployments, where applications became more modular and discreet, with deployments happening monthly, weekly, multiple times per day and even hundreds of times per day. We went from small-scale applications running inside of corporate firewalls to web-scale applications that can serve millions of requests per second on distributed infrastructure.

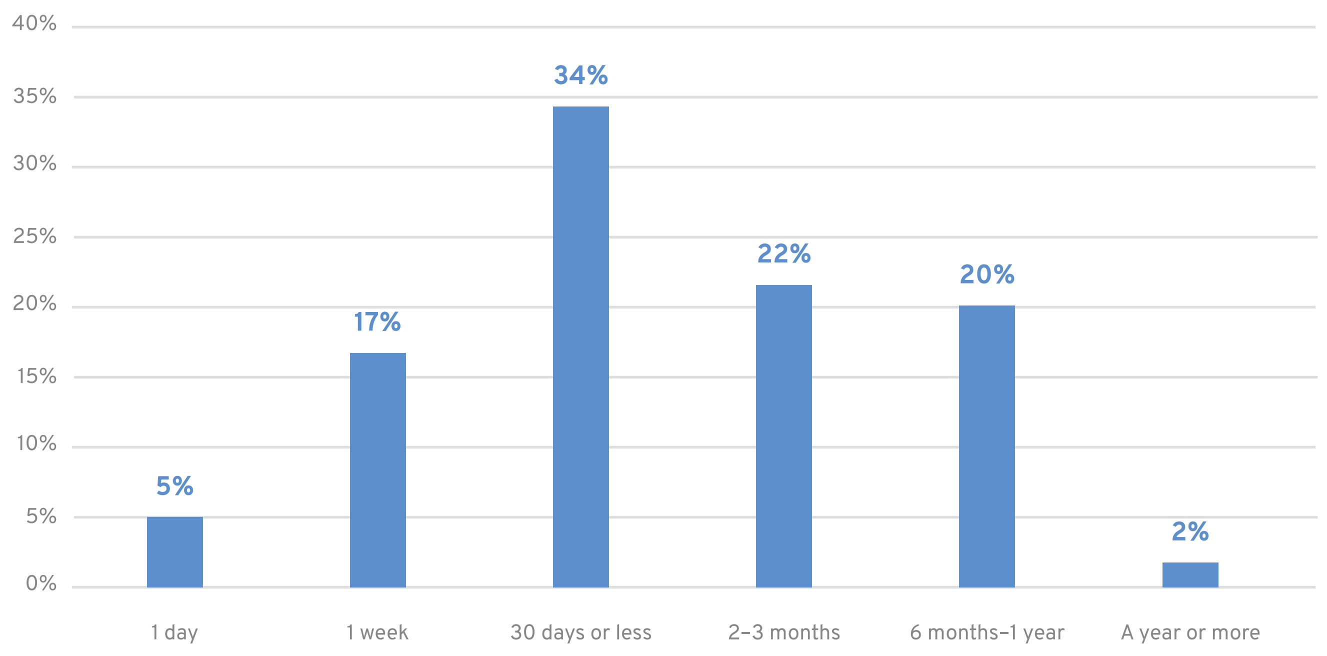
That evolution in speed came through a virtuous loop of new strategies, new tools, more mature languages, and better libraries, as well as more robust serving and scaling frameworks. That loop includes a well-known and highly effective strategy of rolling out code, testing it, rolling it back, serving requests and scaling it to meet surging requests, as well as the use of software tools to support serving and scaling.

While AI/ML deployment builds on many of those foundations, AI/ML models also present some very different factors that currently make them more challenging to deploy and scale.

In 2020, Algorithmia (now DataRobot), found that only 14% of organizations were able to deploy a model in 0–7 days. They reported that 28% of organizations took 8–30 days to deploy a model, 22% took 31–90 days and 13% took 91–365 days.

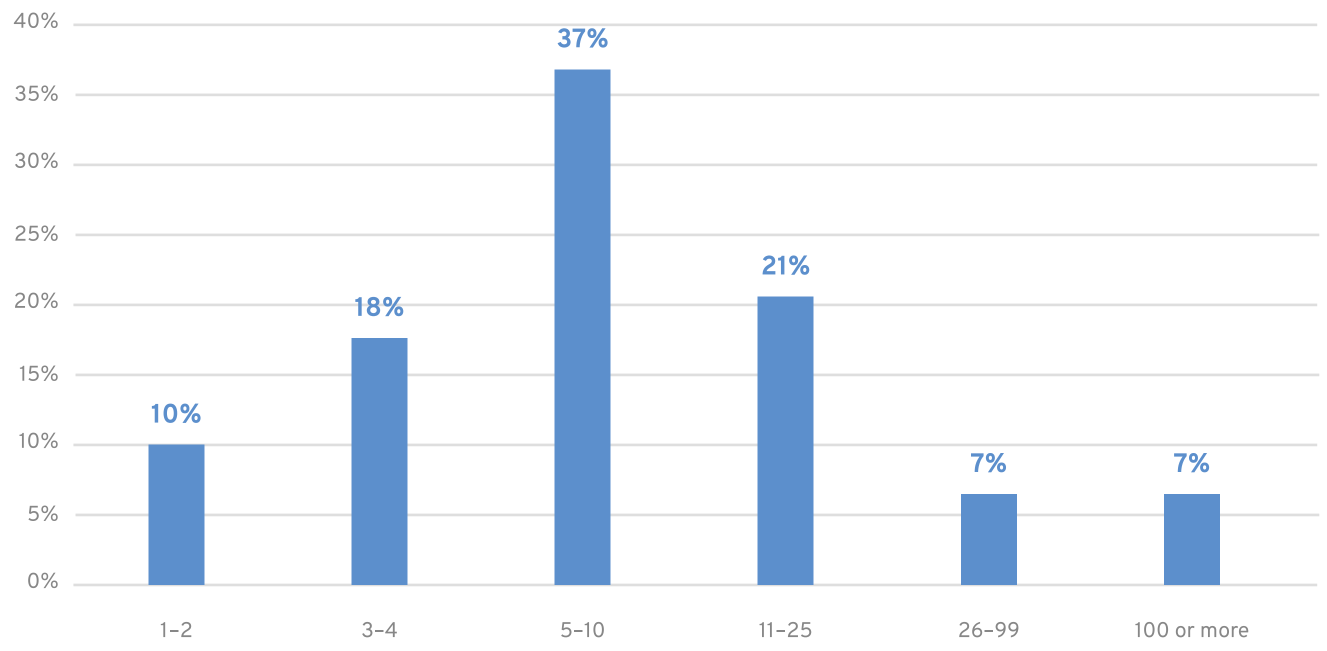
Two years later, in 2022, our survey shows that the numbers have changed. Teams are getting better and faster at getting models into production, but many are still struggling. Only 5% were able to get a model into production in a day, with 17% taking a week or less and 34% getting models into production in 30 days or less. That said, a significant portion, 23%, were still taking 2–3 months to get models deployed, and 20% were still taking 6 months to a year.

How long does it take to deploy a model to production?



Teams are also deploying and managing a growing number of models in production, with 37% managing between 5–10, 21% managing 11–25, 7% managing 26–99 and 7% managing a 100 or more.

How many models are you putting into production per year?

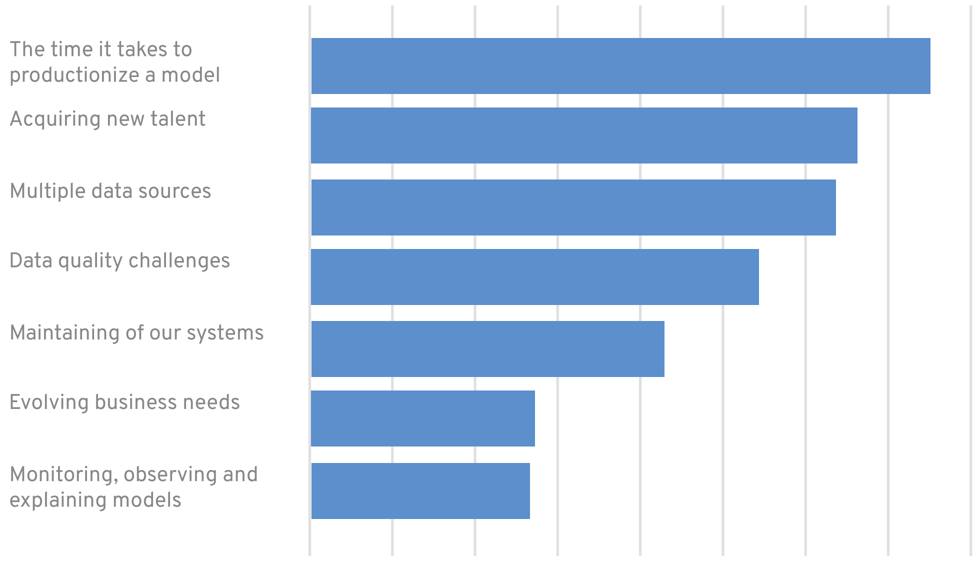


That means it's time to finally put the often quoted stat that 87% of models never make it into production to rest for good. It's just not true anymore.

Of course, there is still room for improvement, but MLOps teams are starting to match their DevOps counterparts in terms of speed and skill. Google found in their 2021 annual survey, The State of DevOps, which tracks traditional application deployment and serving, that the majority of the teams they surveyed were elite or high-performing teams. They found that 26% of teams surveyed hit elite status, meaning they were able to deploy applications on demand or multiple times per day, versus only 7% in 2018. Furthermore, 40% of teams hit "high" status, meaning they were able to deploy once per week or at least once a month. Twenty-two percent of MLOps teams can deploy a model in a week or less and 34% can do it in 30 days or less, which means many MLOps teams would qualify for a "high" status.

But with all that said, teams still say the hardest thing to plan for when building and operationalizing models is the time it takes to deploy a model.

What are the hardest things to plan for when building and operationalizing your AI/ML infrastructure?



When we talk about model deployment and serving we're really talking about two things:

1. **Deployment:** The workflow to get a trained model into production
2. **Serving:** The platform that hosts and serves the model to respond to requests

The workflow involves both the human workflow and/or an automated or partially automated series of steps to package up a trained model and get it into production, as well as the software tools to support the workflow.

When it comes to serving the model, we're talking about a platform to receive and respond to requests with predictions and inference, as well as the software tools to maintain, manage and monitor those platforms.

Typically the companies looking to assist teams with this part of the MLOps landscape are looking to help your team handle both the deployment workflow and to serve the model at scale.

Deployment Workflows

The workflow consists of all the steps that are involved in taking a trained model and getting it into production. That might include creating an instance of the model that can receive requests for inference, packaging up all of the specific runtime libraries in either a proprietary format or a container like Docker, defining ports and opening the firewall, running manual and automated tests, security checks, pushing the model version to a version tracking databases, running regression style tests, spinning up instances of the model to receive requests and more.

The speed that a model gets through this workflow depends on how much of the pipeline is automated versus manual. It should come as no surprise that teams relying on a lot of manual steps are further down the speed-to-production curve, taking months or even a year to get a model into production. Teams that have a high degree of existing automation in the web applications and in-house applications are often better suited to carry that knowledge over to machine learning.

Every company surveyed here offers some set of workflow wizards and automation steps to help speed through the process of packaging up, versioning and deploying a model to a serving framework, including Seldon, DataRobot, Iguazio, Modzy, ClearML, OctoML, BentoML and UbiOps. The workflow tooling platforms use either their own proprietary wizard-like steps, or they combine their tools with external CI/CD tools like Git. They also typically leverage open-source package management solutions like Docker, though some support bare-metal deployments, and some, like BentoML offer their own unified model packaging format for deployment. Some companies, like LGN, offer a deployment workflow specifically geared to AI at the edge, such as on a smartphone, which has special considerations such as memory, and speed and size constraints.

Serving

After deployment comes serving, where a model serves requests for inference. This is similar to a web application in that it needs to receive, queue and respond to requests quickly. AI/ML models typically have higher compute and memory requirements versus traditional web applications, although there are exceptions with memory-hungry languages like Java.

There are a number of key characteristics of a good model-serving framework:

- Framework agnostic
- Scalable

- Replicable
- Request queuing, batching and load balancing
- High concurrency and low latency
- Able to support GPU, TPU or other AI-accelerating hardware

We'll highlight how each of these appear in a typical model serving architecture as we describe the architecture below.

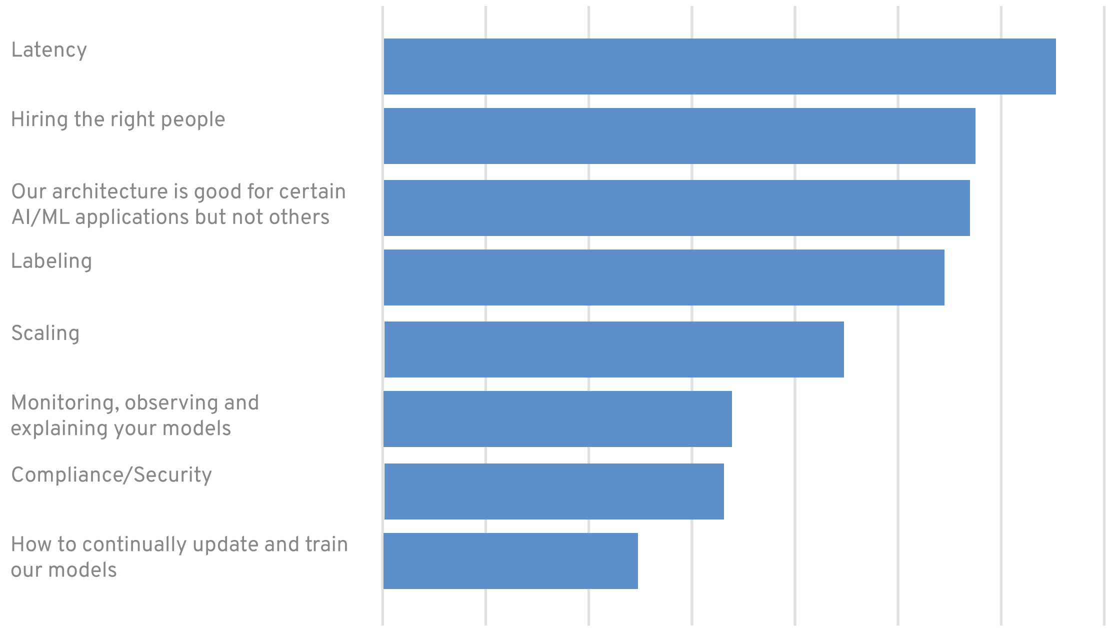
Generally a model server takes a model "endpoint," which is an instance of the model that is able to process and return inference requests. Requests may come from external applications via REST, gRPC or HTTP/HTTPS or from a message streaming service such as Kafka or RabbitMQ. Requests are often queued up or batched by a scheduler, which routes the requests back to the various instances of the model. The scheduling queue connects to the model agent, which in turn runs AI/ML frameworks, such as TensorFlow or Pytorch, or custom-built frameworks that deliver an inference response. The response is funneled back via HTTP/HTTPS, REST or RPC.

Sometimes the scheduler is limited to single instances of the model endpoint, in which case a deployment would need a software load balancer to route traffic across instances. In other platforms, the scheduler acts like a load balancer itself, intelligently routing to backend instances of the model endpoint.

Companies cited latency as the number one concern in production. This differs dramatically from in academia, where teams publishing a paper are looking to maximize their state-of-the-art results and throughput. In academia, teams don't much care if their model doesn't respond swiftly or uses a lot of memory, but in production, latency really matters.

If the ML model that powers Gmail's suggested sentence completion doesn't respond faster than you can think of or type that sentence, then the model is largely worthless.

What are the most significant challenges you face when building your AI/ML infrastructure?



The primary architectures of serving a model are the following:

- Web service
- Containers and microservices
- Serverless

The first approach is to wrap the model up as a web service, with something like Flask, which works well for Python-based models.

The second approach is to serve it as a container or series of containers, such as in a Kubernetes pod.

The last approach is to serve it as a microservice like OpenWisk or Nuclio or Fission, which is essentially a very thin container, and route requests to it via a service mesh like Istio.

Scaling of models is generally done via one of the following:

- A proprietary or open-source clustering mechanism
- Containers and microservices controlled by an orchestrator such as Kubernetes
- Serverless platforms

An example of proprietary or open-source clustering mechanisms are Spark clusters, which can pull from the MLflow model registry, which is a database of model versions. However, Spark is generally not used for serving, and Databricks considers serving via MLflow a public preview and recommends it only for “low throughput and non-critical applications.” It is also currently limited to Conda-based Python applications only, which means it’s not framework agnostic.

Another popular open-source autoscaling and clustering framework came out of the Rise Lab at Berkley and spun off into a company that extends and maintains the framework, Anyscale, which offers a managed services cloud-based version of Ray. Ray is a compute framework for distributed machine-learning applications, as we discussed earlier, and it is not an alternative to containers or microservices. Instead, when it comes to serving, it is similar to something like Dask, which offers a framework for building distributed analytics applications in Python.

KServe is one of the more well-known and widely used open source serving frameworks. It creates Custom Resource Definitions for ML models on arbitrary frameworks. It was developed jointly with contributions from NVIDIA, Google, Bloomberg, IBM, and Seldon as a cloud native model server for Kubernetes. Recently, it graduated from the Kubeflow project as a standalone project. It can be deployed on Kubernetes or serverlessly. For serverless, the system uses Knative Serving, which bakes in automatic scale-up and scale-down capabilities. It uses Istio to expose service endpoints to the API. It can support canary and blue/green deployments and supports a good range of ML frameworks. It can also work with existing serving engines like TorchServe, Tensorflow serving and Triton. It can host PyTorch, TensorFlow, and ONNX runtimes through Triton. It can also serve XGBoost and SKLearn through Seldon’s MLServer.

Most clustering mechanisms can also use Kubernetes or Slurm as a compute substrate, in essence acting as an orchestrator of orchestrators. Of course, the most well-known open-source clustering and orchestration platform is Kubernetes, and we’ve found that most of the frameworks here utilize Kubernetes or support it at some level for orchestration and scaling.

We’ve also found that most serving engines are agnostic to the AI/ML framework or support a wide variety of frameworks. Typically this accomplished by packaging the AI/ML framework up in a container. Framework agnosticism or wide support for frameworks is crucial because there has been a massive proliferation of open-source tools that data scientists leverage to train and build models, including, but not limited to, Pytorch, Tensorflow, ScikitLearn, Caffee, Theano and XGBoost. There are also pre-trained models available, like HuggingFace’s transformers library, which includes foundational models like BERT, CLIP and Perceiver IO, which teams can fine tune for their own tasks. A conversion system like ONNX, which creates a common file format to represent machine-learning models, may help, but most teams surveyed ended up running their models in their native format to avoid challenges with conversions or performance issues. Teams may use a framework to compress or optimize the model before serving it, as with OctoML’s compression service based on the open-source TVM project, in order to lower latency, memory and compute requirements.

As of the time of our survey, the majority of large-scale, pioneering AI/ML organizations like Google, Uber, Lyft and Netflix, as well as the top AI/ML research organizations like OpenAI and DeepMind, use their own custom deployment workflows and tools, typically built around a collection of in-house proprietary and/or open tools, like Git, Docker and Kubernetes, which they saw as an extension of their existing automated CI/CD process.

This is typical in any early stage software ecosystem because when the pioneers start to build their applications, there are simply not commercial or open frameworks to build on, so they have to roll their own. Over time, as the market develops, commercial vendors learn from the successes and failures of the pioneers and create stable software that gets adopted by the next set of organizations and enterprises. These organizations aren't usually research organizations, but they have the staff and know-how to implement the research in their own environments and they begin to leverage open and proprietary tooling to advance state-of-the-art technology.

The vast majority of enterprises today, outside of the pioneering tech and research organizations, leverage one of three kinds of tools: open source, enterprise versions of open source or proprietary serving engines. Open-source tools like Seldon Core or Modzy's Chassis are usually supported via their in-house IT operations on the existing Kubernetes platforms they already run for traditional web applications. Organizations often turn to commercial vendors that offer enhanced versions of open source tools, such as Seldon Deploy or Anyscale Ray's managed service or proprietary serving frameworks like Algorithmia (now part of DataRobot's AI Cloud).

Most of the model-serving commercial platforms like Modzy, Seldon or DataRobot's AI Cloud either support a wide variety of frameworks or they are framework agnostic.

Scaling involves creating multiple instances of a model and routing traffic across the instances to handle large volumes or concurrent requests. Scaling on almost every platform surveyed is done via containers running on Kubernetes or on a dedicated microservices architecture, which is essentially very thin containers and which still often runs on Kubernetes and containers on the backend. Examples of serverless platforms used for model endpoint serving and scaling are Nuclio, Amazon's Lambda and OpenWhisk, which runs on Kubernetes and OpenShift. Examples of microservices are Algorithmia's (now DataRobot) serving platform, Microsoft Azure's Kubernetes service and Azure Service Fabric. The vast majority of these platforms are managed versions of Kubernetes and containers with extensions to make it easier for them to manage the Kubernetes clusters.

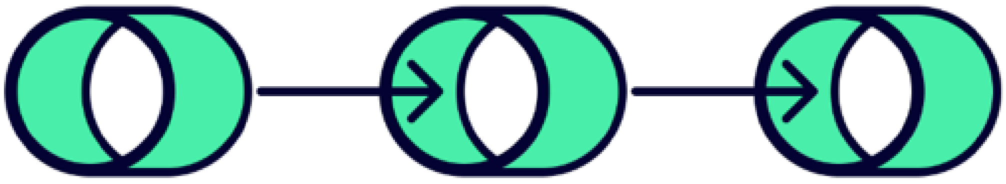
Beyond scaling models, we've noticed that many of the platforms support advanced deployment scenarios, such as **canary** and **shadow** deployments. It's worth checking the support for advanced deployments with every vendor you're considering for your deployment and serving needs. Shadow and canary deployments in AI/ML differ from traditional applications, and it's worth digging to discover whether your vendor understands those differences. For example, a shadow deployment, where requests are sent to the current production version of a model and the test version of the model, may need to persist for a lot longer than a web application because it may take a longer time to establish whether the model's inference is improved or degraded.

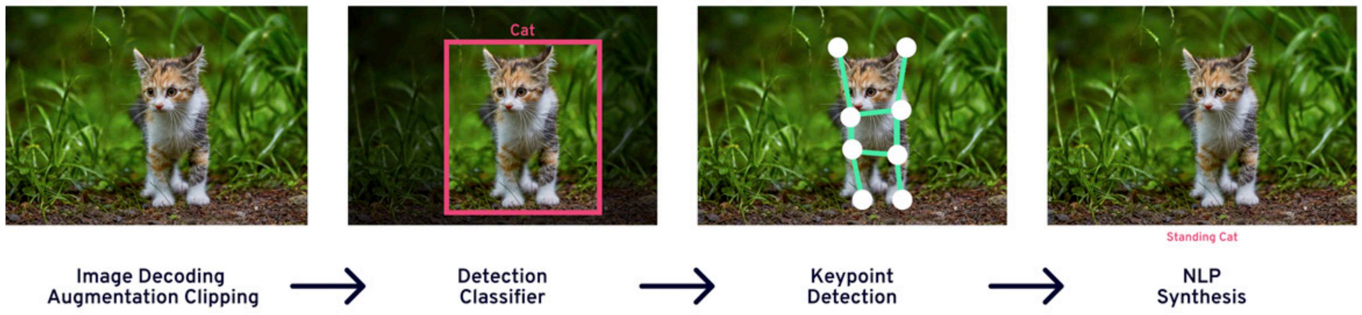
Patterns of Deployment

Lastly, while we often refer to model deployment as singular, as if models were monolithic, typical production deployments are often a series of multiple models or transformation stages, along with traditional business logic to support those models. It's best to think of deployments as deployments of a cluster of models and business logic that forms a complete application.

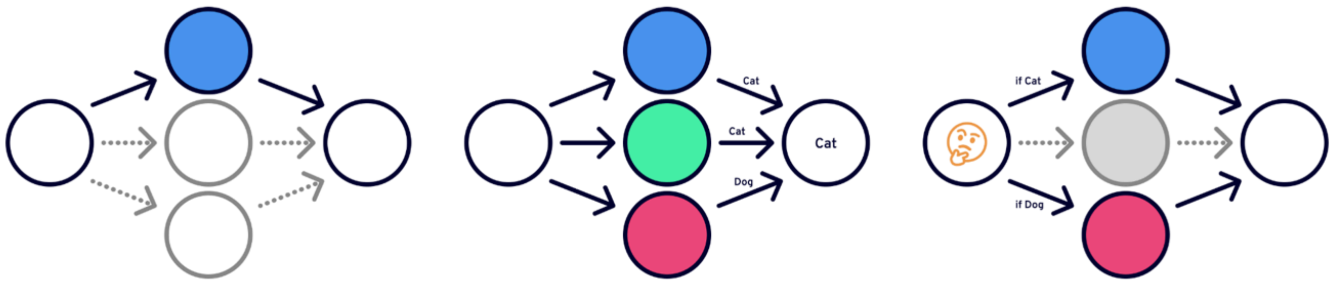
The team at Anyscale noted at least four different common model deployment patterns that each come with their own difficulties. While it's not an exhaustive list, we've found that it largely matches the typical production deployments today. Each type of model presents its own problems in production, and it's worth knowing whether your vendor has a detailed understanding of these kinds of deployments.

The first is the pipeline-style deployment. Pipelines go through a series of linked stages. A request passes through various algorithms or models like beads on a string. A good example is something like a movie recommendation engine, where an inference request passes through various stages like an embeddings lookup, feature interaction, nearest neighbors and rankings before returning its result.





Ensemble models are a collection of upstream models that receive the same request and all return an answer to the requester. That could be for a number of reasons. The first is that you may have a newer version of a model running in a shadow deployment. That means a newer version of the model is live and receiving traffic, but you want to make sure that the model is providing good results, so the older model remains primary and acts as a baseline until the shadow model is ready to take over as the primary.



The second is to give aggregate answers, which can decrease inconsistencies across different models or different versions of a model. A final request may simply take the average of the answers or select one of the answers to test its response with users.

Lastly, dynamic selection may route requests to different models based on the model's individual specialization. A pet owner might indicate they own a dog on a web form, and the service would route to the model that specializes in dogs instead of the model that specializes in cats.

The third pattern is **business logic**. A model is usually surrounded by traditional coded business logic and rarely exists independently. That means traditional application code needs to interface with the model.

The last pattern is **reinforcement learning**. Reinforcement learning (RL) is unique in that an agent endpoint is making decisions and receiving a reward or punishment for its decision-making. This is the most advanced style of deployment, and for now it is largely confined to the most advanced research houses and large-scale tech and financial companies as it requires real-time data feeds and tremendous scalability. Ray is one of the few platforms built to handle RL.

All of the patterns present challenges for scaling, and the top platforms provide fast interconnects between nodes and endpoints, fault tolerance to address non-responsive or downed nodes, and error correction. Each of these styles of models has its own challenges for maintenance and uptime, and require excellent traditional IT monitoring and management. For example, a pipeline model may fail at any stage, and if any stage fails, the entire output fails. An ensemble model may be more forgiving if it is simply taking the aggregate of answers from all the models, but if the failure persists, then the results are coming from a reduced set of models and thus suffering from the weakness in the individual models that the ensemble was looking to overcome. In the monitoring section of this report, we'll cover the kinds of monitoring typically found in modern AI/ML production deployments to ensure that models stay running smoothly.

Current Trends and the Next Five Years

Deployment and serving are some of the most challenging and yet the most important pieces in the AI/ML puzzle. Too often, teams find themselves struggling to productionize machine-learning models after they were developed with a wide variety of open-source, homespun and proprietary tools. Memory issues, the security of the data and models, inference latency, dependencies, software versions, scaling and a thousand other factors can hurt performance. Drift can make a model useless in a short period of time, and teams need a smooth way to get models into production. Monitoring, as we'll see in a later section, requires different tools and different metrics than a typical web monitoring setup, requiring teams to familiarize themselves with additional platforms and concepts.

There is a wide range of deployment assistance in almost every product surveyed. We recommend every team look carefully at the deployment process on any platform they are considering and take a close look at how it works with a variety of simple and complex model deployments, from single models to pipeline models to ensemble models. Ask questions, such as what frameworks are supported? Is the process agnostic? How many methods of rollout are supported and how do they work?

When it comes to serving models, we are seeing a split in approaches. The dominant approach is to containerize the model and serve it with all its dependencies and code and then replicate the container. This is the most popular approach and the most flexible. The reason for this is simple. There are thousands of libraries and dozens of major frameworks in multiple languages. Python is the most dominant, but apps that use R, Scala or Java are important too, especially in areas like fraud prevention, which has a long history in enterprises that use more than just Python. Containers are the most flexible way to deal with that diversity.

The second approach is to build a framework in a chosen language, usually Python. This is the approach Ray and Dask have taken, and it's useful in that it goes where most machine-learning libraries currently exist and leverages that to allow people to build scalable, distributed machine-learning apps without altering much or any of their code.

However, we recommend teams look at serving as being three layered. The first layer is an underlying general purpose compute, such as Kubernetes. The second layer is the leveraging of a serving engine like Seldon that uses something like Kubernetes but adds in paradigms and functions for machine learning. The third layer is a framework that runs atop either a first or second layer solution like Ray, Dask or Flask.

It's essential to know where each layer fits into the picture and which ones are dependent on other layers to function. Each layer adds a level of complexity to management but may also offer significant advantages like the autoscaling or sunsetting of old models or making it simple to package models in a consistent way for running on the web or on an edge device.

The Next Five Years

In the current generation of deployers, container-style deployments such as Modzy, Seldon or Algorithmia (now DataRobot) are the most dominant choice, and we largely expect that trend to continue. It is likely to prove too challenging for any one team to have to continually support the massive proliferation of formats individually in the long run. It's simply too complex and too error prone. Containers provide an elegant solution to the problem in that the platform does not need to understand every language or framework explicitly, though some frameworks do have unique dependencies, so the problem is not totally eliminated. Serverless frameworks like OpenFaas, OpenWisk, Fission or Nuclio that sit atop containers also play a role by stripping containers down to their minimum and enabling fast and fluid microservice meshes of models.

It's also possible we will see the rise of a purpose-built container interface format for models, such as what the MLCommons team is working on with MLCube, though it is likely that such a format will remain on a general-purpose container as well and simply augment it. Think of it as a container within a container or as a series of connectors between containers.

We have already seen attempts to build a standard model format with ONNX, and that is a solid addition to the field, but we continue to expect most teams to run their models in their native format, as the conversion process can sometimes be problematic or introduce additional complexity. The advantage of ONNX is that converting to the format allows you to avoid environment dependencies like requiring the Python interpreter and its libraries, as well as various version conflicts. ONNX stores both the architecture and the parameters in a single file, and that helps simplify deployments on a serving engine. The main challenge with something like ONNX is that the ONNX protocol or the converter script for your particular model may not support all of the operations of the source model.

We also expect to see the rise of multiple compression and conversion systems to shrink the size of models. We are already seeing that with platforms like OctoML and their Apache TVM compiler and LGN with their Ultra platform that does model pruning and compression to make it run smoothly on edge devices. This will become increasingly important as we see more and more models deployed to memory constrained edge devices and phones. Many models today are trained with absolutely no consideration as to their size and memory footprint because they are expected to run in a datacenter or the cloud. But more and more, AI models will advance to running on smartphones and the coming generation of augmented reality (AR) glasses, and they will need to fit into a much smaller footprint. It's simply not possible to run many giant, memory-hungry models that work in a datacenter or the cloud on a phone, and we expect more compression and conversion frameworks to optimize models.

We do expect frameworks like Dask and Ray, which enable more advanced use cases in machine learning, to gain traction, but those frameworks will still need to sit atop a general-purpose serving engine.

The serving space was one of the first to get funding from venture capital, and it has a large variety of players, behind only orchestrators and pipelines in terms of the number of total platforms on the market. We expect more and more consolidation in the serving space over the next five years, and in ten we expect the market to have largely consolidated down to a small subset of major players, just as we saw the web serving market go from fifty or more platforms to two or three dominant ones.

Finally, we don't expect there to be much consolidation on the "deployment" side of the equation, as platforms that speed up the complex process of packaging up frameworks are more akin to wizards from the early days of computing. They help simplify a difficult challenge, and there are a number of ways to do that. It's likely many platforms will continue to offer deployment as part of their overall platform to speed the path to production, and we don't expect there to be a standardized way to do that in the near future, if ever.

AI SUPERVISION: MONITORING, OBSERVABILITY AND EXPLAINABILITY

Companies and platforms covered in this section include:

Arize, WhyLabs, Fiddler, New Relic, Qualdo, Aporia, Arthur, Evidently, Seldon, Dataiku, Modzy, Mona, Censius, Amazon SageMaker, Google Vertex, Azure Machine Learning, DataRobot, Domino Data Labs, Prometheus, Grafana, Acceldata, TruEra, Zabbix, Bosch AIShield.

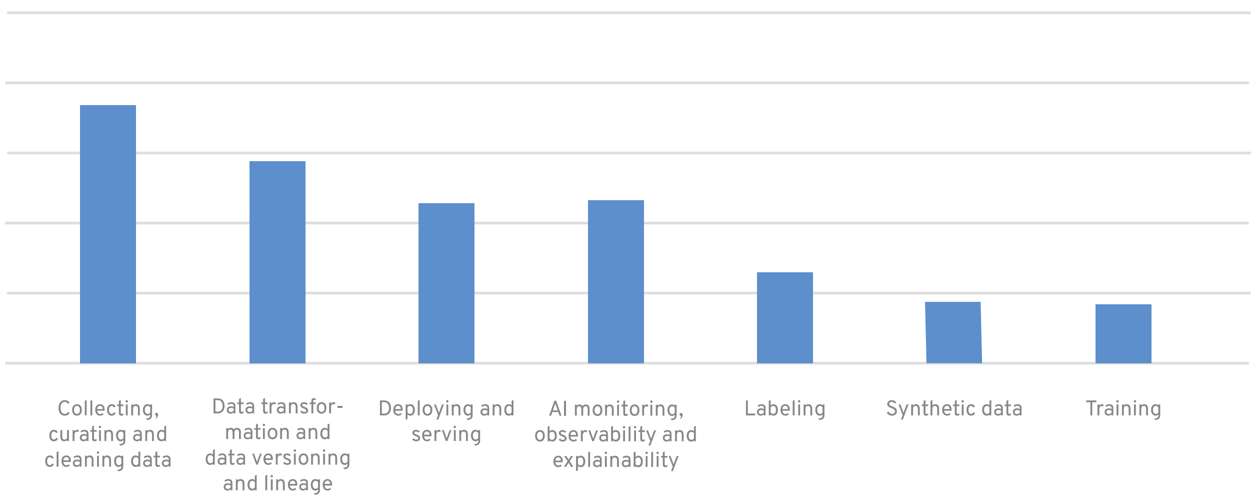
Introduction

Monitoring and observability have a long history in IT, with companies like Solarwinds, Datadog, Splunk and Dynatrace delivering robust cloud offerings for monitoring and managing IT infrastructure. Open-source tools play a big role in monitoring as well, most notably Prometheus, Grafana and Zabbix, which monitor everything from Kubernetes clusters to databases, networking and web applications, and they can be deployed on premises or as a cloud service.

AI/ML monitoring, security and observability are relatively new entries in the space, and they have their own unique requirements and challenges. AI/ML security is the newest entry in the space, and it is a subset of observability and/or explainability. In particular, they often require a different backend than traditional IT monitoring engines. They also require companies to collect different kinds of data.

According to our survey, the space is tied for third place in a ranking of the areas that teams expect will have the biggest impact on their businesses in the coming years. At the AIIA, we think organizations will start to rate it higher as they deal with increased regulation and the challenge of managing a fleet of misbehaving models in production.

Where have you faced the biggest challenges in productionizing models?

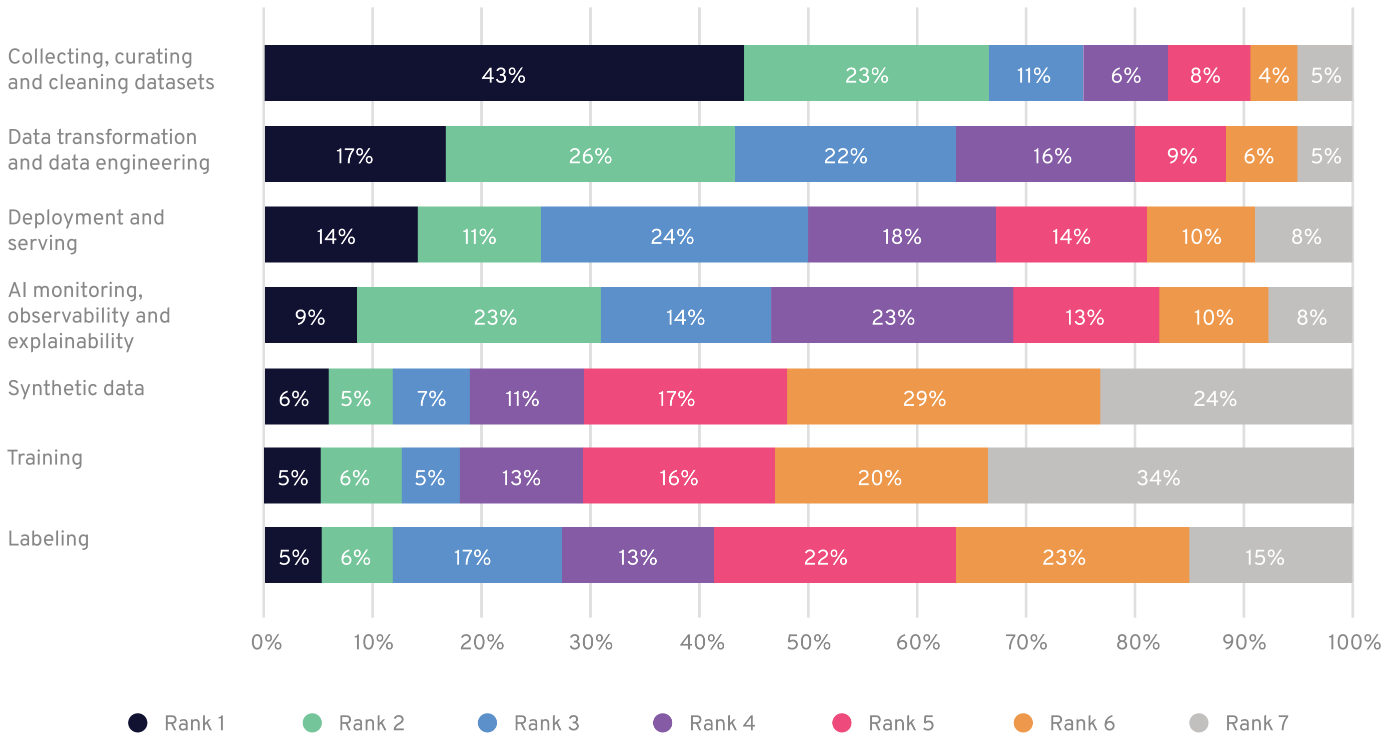


Not only that, but it also comes in tied at third for where teams feel they need to invest the most talent, time and money, trailing data-centric and data engineering steps, and deployment and serving.

You should not think of AI/ML monitoring and observability as the same thing as traditional IT monitoring, due to its unique characteristics, which we discuss in detail in this section of the report. Until your monitoring provider explicitly offers AI/ML monitoring, you can safely assume that their tools are good for monitoring the traditional characteristics of an application like uptime, memory, CPU usage and the like but not the specific characteristics of AI/ML models like inference and prediction accuracy, drift and data quality.

In this report, we are coining a new overarching term to encapsulate the capabilities of AI operations and to reduce confusion, since monitoring is a general-purpose term in IT that also has a specific meaning in AI/ML. That term is **AI supervision**. Supervision includes **monitoring**, **observability** and **explainability**, and those three terms touch everything from the pipelines to the feature databases to the models to the data itself. We explain the key differences between the three terms in the next subsection, Overview of Monitoring, Observability and Explainability.

What parts of the AI/ML infrastructure should receive the most resources (i.e., talent, time, money)? Rank in order from most to least. (Ranked first)



You should also be careful not to confuse a monitoring engine that uses AI on its own backend to augment its own traditional monitoring engine, such as Splunk's Machine Learning Toolkit or Datadog's machine-learning backend, with AI/ML model and data monitoring. That's often called AIOps, and it's not what we are talking about here. Many application providers use ML to help make predictive recommendations to their customers about potential failures of IT systems, using techniques like anomaly detection, and to alert systems administrators of potential problems before they happen. Anomaly detection signals when data is suddenly outside its normal range, such as when electrical usage shoots up or down suddenly after staying steady for weeks or months.

We can also loop in **security of data and models** here, under the umbrella of AI supervision. Like any IT system, AI systems can be vulnerable to cybersecurity threats, and their protection depends on the security measures deployed across the system. While current security engineering guidelines and practices should be followed when deploying AI, organizations also need to address the rise of new AI security threats like extraction of the model, poisoned training data, leakage of training data, the malfunctioning of the system through manipulated inputs and evasion of the intended functionality. We're also facing novel threats using newer attack surfaces like data, training, framework libraries and the logic of the model itself. Withstanding these new attacks requires novel system design and defense strategies.

AI/ML also involves a unique type of monitoring called **explainability** that is not found in traditional IT monitoring platforms. This is due to the unique nature of machine-learning models, which are often referred to as **black boxes** because we don't exactly know why they make the predictions they make, unlike with handcrafted code logic, where we can examine the code to see why an application failed or succeeded. Explainability seeks to crack open the black box and give auditors insight into why an ML model made a decision.

Overview of Monitoring, Observability and Explainability

Typically, there are five key stages to monitoring platforms:

1. Gathering of data
2. Data preprocessing and data engineering
3. Running analytics
4. Displaying the output via text/graphing/visualizations
5. Alerting

A sixth optional stage may include automated remediation, where the system takes automatic steps to try to correct a problem, such as with Red Hat's Insight's platform, which uses pre-defined Ansible scripts for programmatic remediation, Datadog's webhooks (not to be confused with DataRobot), which allows faults to trigger bespoke code, or Bosch AIShield, a defense engine that can take predetermined steps to thwart security incidents.

There are three kinds of supervision of AI/ML models and data:

1. Monitoring

2. Observability
3. Explainability

The terms **monitoring** and **observability** are often used interchangeably by marketing teams, which creates confusion. While there is a lot of overlap, there are some major differences. Sometimes **explainability** is used interchangeably as well, but it should be considered a completely separate subset in the space, and most companies and platforms refer to it as a subset that stands on its own.

Typically, **monitoring** tries to answer the questions of what and when. Is a web server up or down? When did it come up and when did it go down? Monitoring capabilities track most closely with traditional IT monitoring and include uptime, performance and the like.

Observability gives teams context on how and why. The model went down because the last deployment created instabilities that caused it to crash after ten minutes. It helps answer questions like why is the model's prediction performance degrading?

Explainability is achieved with a suite of algorithms that help humans understand why a model made a decision after the fact or what a model is focusing on when it makes decisions.

In general, an easy way to understand the difference is that monitoring tracks failures, outages, uptime and performance, while observability helps us to understand the system in both a healthy and unhealthy state, and explainability answers questions about specific predictions or inferences or the model's focus as a whole.

Lastly, there are two areas that AI/ML supervision platforms tend to focus on:

1. Model supervision
2. Data quality

When it comes to model supervision, explainability tends to apply almost exclusively to models and their performance after training or when they reach production, but both monitoring and observability need to exist for the ML system as a whole, not just for the model. A bug introduced into the data pipeline may cause a model to fail, but a team needs to be able to track the problem back to where it started in order to fix it, which means looking at the whole pipeline.

Data quality is a subset of supervision. It focuses on testing and evaluating data as it comes into the system. It looks to identify problems such as missing data, out of range violations, type mismatches and more.

Survey of the Field

Since this is a relatively new field, many large platforms are still building out their offerings, leaving it to startups like Arize, Fiddler, Aporia, WhyLabs, Modzy, InfuseAI, Qualdo, Arthur, Evidently, DataRobot, Superwise and TruEra to fill in the gap. We're also beginning to see large traditional monitoring platforms move into the space, such as New Relic.

By contrast, the large cloud providers, which offer AI/ML capabilities that are better thought of as a suite of tools rather than a single unified platform, have been slow to offer monitoring, observability and explainability. Azure Machine Learning and Dataiku have only a few capabilities in this area, such as a preview/beta drift detector in Azure's case. Amazon SageMaker offers data quality checks, drift detection and some bias detection. Dataiku does have a product specifically focused on explainability, one of the few large providers to have one at this point. Google cloud offers only training-prod skew and drift detection with their Vertex AI platform.

Companies that address monitoring, observability and explainability fall into a broad range of categories. The first category is companies whose primary focus is AI/ML monitoring, observability and explainability, or who have an entire product suite dedicated to all three capabilities. These include Arize, Fiddler, TruEra, Aporia, WhyLabs, Superwise, Qualdo, New Relic, Mona, Censius, Arthur, Evidently and DataRobot. It is worth noting that DataRobot's marketing team refers to all AI/ML monitoring capabilities as explainability, but we will stick to the industry-standard usage of the term here.

Each of these companies brings a slightly different approach to the way they do monitoring, observability and explainability, while also offering similar, standard capabilities like dashboards and visualizations.

For example, Superwise looks to move beyond just dashboards and visualizations and has a strong and flexible policy creation framework. Their policy builder lets companies express the kinds of things they want the platform to scan and monitor on an ongoing basis. Those policies feed into their incidents system, which automatically groups events that violated those policies so that teams can perform better root-cause analysis.

Arize offers a SaaS and on-prem solution that natively supports tabular/structured data types (strings, floats, booleans, etc.) and they are one of the first to support NLP, image and other unstructured data types. Arize has a robust, clean and clear dashboarding system and a strong focus on observability and surfacing potential problems. For example, a potential problem could involve a credit worthiness model that's overly focused on a particular feature that might cause compliance challenges or miss important creditworthy customers. Arize's developers focused directly on problems data scientists experience regularly, in addition to providing detailed bias and fairness checks, along with scalable post-production monitoring and alerting.

Fiddler provides excellent dashboards and alerting, along with strong role-based access control, which is key for security in enterprises. Its API lets you send or receive data from the system, which allows for easy integration into third-party tools like notebooks. Its dashboards offer low-code analysis, collaboration and human-readable explainability. It also provides unique compliance-level checks like fairness, which can detect whether models violate the Equal Employment Opportunity Commission (EEOC) commission's twelve factors of discrimination.

Aporia delivers self-hosted deployments that can run on Kubernetes and the cloud and that differs from a number of the pure SaaS-based solutions profiled here, providing a high-level of security for enterprises that can't use SaaS-based metric systems. It also includes a robust set of integrations for alerting, taking it beyond email alerts to venues such as Slack and Jira, which creates a good deal of flexibility for enterprises looking to triage failing models and make sure they are aware of problems fast.

TruEra provides users with some unique explainability features that are based on six years of research conducted at Carnegie Mellon University. Like Fiddler, it can laser in on model bias for protected categories. It also provides a separate workflow for high-stakes and regulated models that involve a higher level of risk, such as healthcare models, or specific governance challenges. Lastly, it offers easy model comparisons since data scientists train different versions of one model.

WhyLabs works across streaming, batch and real-time modes without any data volume limit. WhyLabs also supports unstructured data like images, though it also handles structured data like the other platforms profiled here. It has a strong focus on privacy, in that the raw data never leaves the customer virtual private cloud, despite it being a SaaS solution. WhyLabs does this by operating with lightweight statistical profiles that get encrypted during transfer and at rest. Lastly, it offers integrations such as DevOps tools (Pagerduty and Slack) for alerting or triggering upstream model training via Webhook and platform-specific integration.

The second category consists of tools that offer a subset of capabilities in addition to their primary feature set. For example, Domino Data Labs has a suite of products but includes extensive drift monitoring capabilities, and Modzy's platform focuses on a range of end-to-end capabilities like training and deployment but also offers drift detection and retraining monitoring. Additionally, Seldon has a primary focus on deployment and serving but offers drift detection and metrics about models that stream to Prometheus and Grafana.

The third category consists of platforms that have a specific focus on explainability as part of their product suite, which includes Dataiku, DataRobot and Seldon, with its Alibi framework. We could also include Bosch's AIShield under observability and/or explainability, as the platform aims to highlight security threats and make them understandable to teams.

Understanding Monitoring, Observability, Explainability and Data Quality Capabilities in AI/ML Systems

We noted that AI/ML supervision requires teams to capture new kinds of data or augment their existing data collection. In this section, we outline why that is and what the capabilities are for different kinds of monitoring, observability and explainability platforms.

Monitoring and Observability

When it comes to collecting the different kinds of data needed to effectively supervise models and data, AI/ML monitoring/observability is often reliant on **actuals**, otherwise known as **ground truth**. Ground truth is the real-life, correct answer rather than the prediction. In essence, models make **predictions** or **inferences** when faced with new data. For instance, consider the following question: is this transaction fraud or legitimate? The ground truth is what actually happened. The transaction was legitimate and made by the person who actually holds the credit card. The difference between ground truth and the prediction indicates a model's performance over time.

In some cases, getting the ground truth is relatively simple and quick. If a model is designed to predict whether people are likely to click on an ad, then the ground truth is collected as the ad is shown and statistics on clicks are collected in sufficient volumes.

In other cases, the ground truth is delayed, and identifying whether the model is performing well is much more challenging. For instance, a model that predicts whether people will default on a loan may not receive data on a default for many months or years. In that case, companies may rely on **ground truth proxies**, which are stats that offer an approximation of the truth. In fraud detection, that might be a false positive or false negative statistic, or with loans it might be the aggregate amount of loan defaults per month based on different demographics rather than the actual individual's loan default statistics.

Lastly, collecting the ground truth may be impossible or require additional applications to collect the relevant data when it comes to something like object detection. Google Photos uses object detection models to identify you, your friends and your pets in photos. But it has no way to automatically collect ground truth statistics because only you know whether the model identified you correctly in a photo or missed you completely in a photo. That's why Google Photos requires a different application to gather that ground truth. It prompts you in the application to audit its inference choices. You may have seen the "help improve Google Photos" prompt in the application, where it asked you whether two different images were the same person. That helps gather feedback and establish the ground truth about the model's success.

Beyond the different kinds of statistics needed for monitoring, observability and explainability, your platform will likely need a different backend as well because traditional supervision engines generally only need to keep track of the current state of a system and don't require much history. It's geared for fast ingestion of current statistics, but historical statistics are less important to the system. For instance, if we are monitoring a web application for uptime, we need to know whether the system is currently up or down so we can take action. We only care about the history of the web service's uptime in the event that it is bouncing or repeatedly going up and down as that time-series data may help us triage the problem. Otherwise, we can safely discard the history after a reasonable time frame.

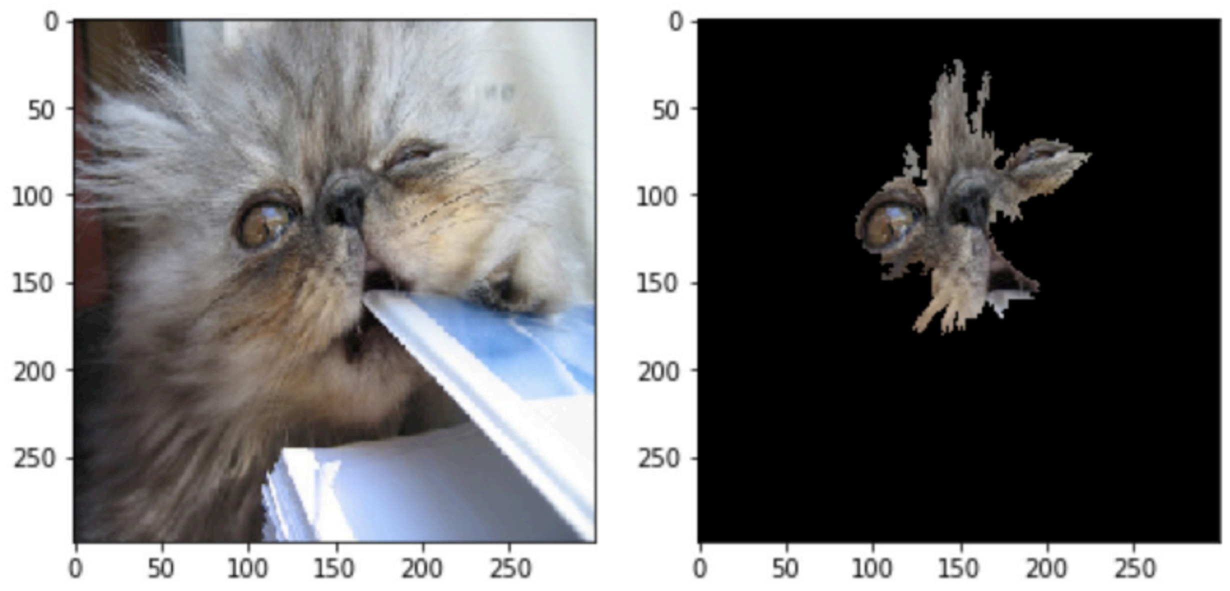
By contrast, monitoring AI/ML systems often requires keeping the entire history of predictions and ground truth answers in order to monitor for phenomenon like **drift**, where a model's inference performance degrades over time. In order to understand drift effectively, we have to continually evaluate the model's inference over time to see if it is degrading with new predictions. Drift refers to when a model is starting to break down because the features of the data have changed. A simple example is a model that recommends clothing ideas to shoppers continuing to recommend fall clothes in spring and causing new purchases to slow down because the shoppers are receiving less relevant suggestions. Another example of drift might come from black swan events like the COVID-19 pandemic, where demand forecast predictions were likely to suddenly fail when lockdowns canceled concerts during their peak season.

There are a number of other types of metrics observability platforms track, such as outlier **detection**, where rare data conditions that throw a model off are kept track of. One of the primary reasons many advanced data science teams look to get big data is because they are really looking for outliers, such as with training self-driving cars, where they capture strange deviations from the norm like a completely blocked-off street, a sudden crash or a street sign covered in stickers that impede object detection. Or **training-prod skew**, which is when a model performs badly on real data after performing well in training and **bias detection**. Bias detection is when you try to understand whether your model is reinforcing its own biases. This may come into play with something like a loan algorithm, where you decide whether someone is creditworthy enough to receive a loan. The problem is that if you reject people, you don't know whether they could have paid back a loan. One approach to solving this problem might be to collect a hold-out set of people you rejected to lend to anyway and then monitor that set for whether the model was correct or wrong.

The classic example of bias in data is this illustration from World War II. Engineers created a heat map of bullet holes in planes that had taken fire in battle, in order to know where to reinforce the plane with better armor. But statistician Ahbrahm Wald noticed that the heat map only included planes that returned and not planes that had crashed, so he reasoned that it was better to put armor on all the places not shown on the heat map because those were likely the most vulnerable spots on the aircraft, which, when hit, caused it to crash and burn.

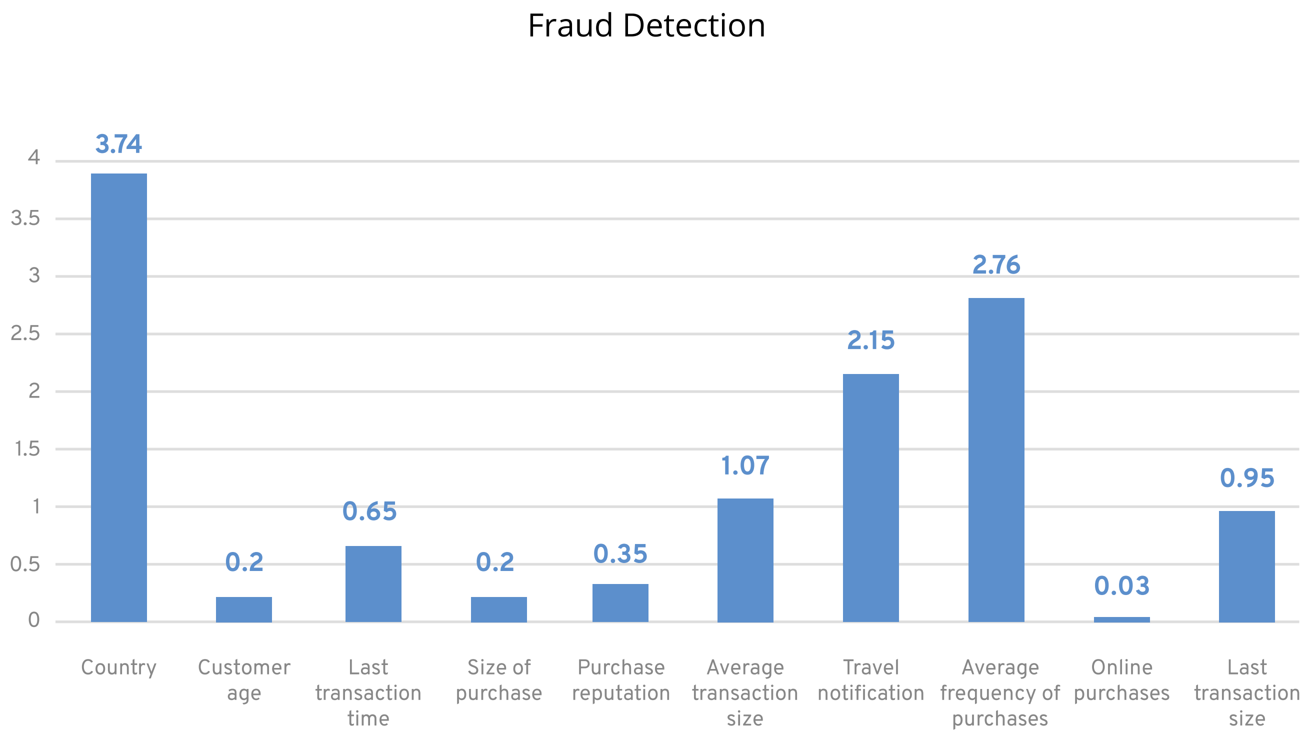
Explainability Explained

Explainability is the final unique piece of the AI/ML supervision puzzle. As noted earlier, explainability is achieved with a suite of algorithms that help humans understand why a black box model made a decision. An example from Seldon's open-source Alibi framework is when an object detection system is queried to show which pixels it focused on to decide an image is a cat, as in the picture below:



Other examples of explainability algorithms might show a human auditor which keywords a model focused on to select a resume from a job candidate pool, or which characteristics of an applicant's profile were most prominent in denying or granting or loan, or which words were highlighted to indicate the sentiment of a support request.

Explainability can be used as a triage tool to fix problems in production models and as a test for models before they make it to production. For instance, a model designed to detect fraud may overly focus on people being out of the country as a key signal. While that may be a good predictor of fraud it may overly penalize highly mobile users or frequent travelers.



We expect explainability platforms to become very important over the next few years and to increase in number substantially over the next five to ten years as regulators and governments increase scrutiny of AI and algorithms. In particular, we expect algorithm auditing tools to become absolute must-haves for enterprises in heavily regulated industries like defense, insurance, drug design and discovery, medicine, finance, banking, telecommunications and vehicle safety.

Data Quality

Lastly, no system is complete without data quality checks. These are more closely aligned with traditional monitoring platform capabilities. Data quality is a somewhat vague term that can cover a number of major and minor issues with data, but for the purposes of this section, we refer to hard failures in data pipelines as data quality issues.

An example of a data quality issue is **missing data**, which is what it sounds like, data that should have come into the system but either didn't arrive or arrived in a corrupted or broken state. Detecting this broken state is critical to saving teams from headaches later, such as realizing several days of GPU training time in the cloud were burned on an incomplete dataset or a dataset with detectable errors.

Data quality issues also include classic errors like **out of range violations**, like when a datum of 75 comes into a set where the range is 0–60, **type mismatches**, like a name listed, and cardinality shifts, where one data steam comes into the system and overwhelms the other streams or where several streams are missing. Examples of **cardinality shifts** might be something like missing inventories of certain products, like umbrellas, in an e-commerce catalog, or an error that includes double or triple the amount of reported umbrellas, which drives the model to incorrectly think umbrellas are under-ordered or over-ordered.

Lastly, data quality control might include deeper checks into the data, perhaps to look for overrepresentation or underrepresentation of samples. For example, a dataset for fraud might include a large amount of legal transactions but a minimal number of actual fraud cases, which makes training for fraud challenging. Another example comes from manufacturing, where a team might have many different widgets that come off the production line perfectly and very few defect samples. An overrepresented sample might be a large number of the same kind of defects, while the sample is missing examples of more serious defects. Advanced data-quality engines will try to have deep insight into whether the data itself has problems and try to surface those to data scientists and data engineers before or during data analysis.

Current Trends and the Next Five Years

AI supervision, with its subsets of monitoring, observability, security and explainability is one of the hottest areas in AI/ML and an area in which a lot of companies and platforms are vying for attention and getting large valuations and investments in venture capital. We suspect that’s because there is a strong historical base of computer science to build on in this area, whereas other areas, like generating excellent synthetic data or training and serving a reinforcement model, are new and sometimes even still experimental. While the backend engines are different and the types of data collection are new for AI/ML, they are still based on a similar foundation to traditional monitoring.

Many of the companies in this space offer similar capabilities, and companies looking to make a decision on which platform to use should consider the following:

1. Ease of use and interface
2. Range of capabilities
3. Ability to support structured and unstructured models and data
4. Scaling capabilities and performance of the platform itself
5. API maturity
6. Customer team and support
7. Ability to easily integrate with other platforms

We recommend that companies closely review the websites of AI/ML monitoring, observability, security and explainability platforms, make a short list, compare features they currently need or anticipate needing in the near future and request a demonstration and then a proof of concept to assist their decision-making.

Monitoring platforms are one of the easier parts of the AI/ML stack to integrate, even with on-prem or homespun AI/ML platforms because they consume data at well-defined end-points from existing components and newly adopted components in a fairly standard way. The rise of successful cloud IT monitoring companies over the last decade has proven the model and created the template for AI/ML monitoring to succeed.

The Next Five Years

We do expect more companies to enter the market in the next five years, and at the same time we expect the consolidation and acquisition of existing players. We also expect that the existing large IT monitoring companies, like Dynatrace and Splunk, will move into the ecosystem, either through creating their own capabilities or through acquisition.

There are also currently two glaring omissions in the space of AI/ML supervision:

- **Security**
- **Auditing and compliance**

Security is one of the newest areas of AI supervision. It's still very much an area of active research. Models and data present new vectors of attack that don't exist in traditional IT systems. For instance, it's possible for a hostile employee to insert poison data into a training set that is imperceptible to a human user, for example, by putting a subtle array of dots over an image to trick a model into seeing something different. It's also possible for external attackers to probe a model to understand its logic and then exploit that logic or get the model to dump its logic to the attacker.

While there are a wide range of tools used to detect attacks in traditional IT environments, such as antivirus signatures, network attack signatures, honeypots and more, there are practically no AI/ML security tools. Bosch AIShield is one of the first platforms to offer protection against a subset of attacks, for example, by detecting whether a model is poisoned. However, because this is still an area of active research, we expect the tools to get much more sophisticated over the next five to ten years as attackers invent new and novel ways to attack AI/ML systems. We also expect the systems to begin to mirror some of the capabilities of traditional IT security, such as automated responses, alerting and visualizations, which help companies laser in on precisely where an attack happened and why.

While many of these tools can be used to ensure auditing and compliance for regulators and government watchdogs—and some platforms such as Acceldata, TruEra, Fiddler and DataRobot note the use of their capabilities to help achieve compliance—we expect auditing and compliance to become a specific subcategory of the supervision space. That's because we expect regulators to focus on the space in the coming years, and we will likely see a maze of regulations that companies need to adhere to and specific styles of reports that they need to generate for auditors. Several platforms, such as DataRobot and Bosch AIShield, already offer exportable reports, and we expect companies will rush to offer exportable reports and to update their existing reports to conform to specific requirements as regulation evolves.

We also expect that many of these reports will include more human readable components, automatically generated to help with compliance needs, which will use ML techniques to help with summarization.

Finally, we expect wide adoption of AI/ML supervision, as few companies can afford to fly blind with data, pipelines, and models in production as they become more reliant on AI/ML models for core business functionality.

TRAINING

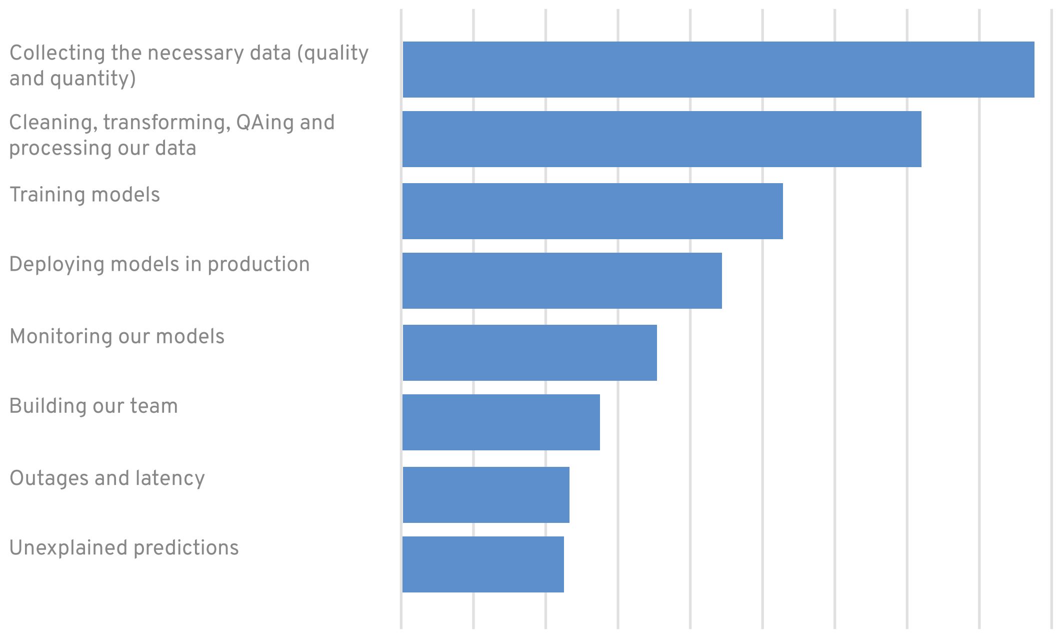
Companies and platforms covered in this section include:

Grid AI, Superb AI, HPE, Google Vertex, Azure Machine Learning, Amazon SageMaker, Apache Spark, BigDL, Horovod, TFX, Pytorch Lightning, Dataiku, Lambda Labs, DataRobot, Juicelabs, NVIDIA SuperPOD, Nvidia DGX, Run AI, ActiveLoop

Introduction

Training is one of the major bottlenecks in machine learning, alongside labeling. Whereas labeling is slow because it requires scaling a distributed human labor team, training is slow because it involves waiting for machines to finish complex calculations. It's also the most systems administration heavy aspect of the AI/ML pipeline. It requires teams to have knowledge of networking, distributed systems, storage, specialized processors like GPUs or TPUs and cloud management systems like Kubernetes and containers.

What components of your AI/ML infrastructure have the biggest impacton your results?



Training ranks third on the list of the biggest challenges companies face when building their infrastructure.

When we talk about training, we're really talking about rapid training or distributed training. In practice, this involves spinning up a series of GPUs or TPUs and parallelizing training across those specialized processors. Of course, not every data science project needs distributed training. But large transformers, language models, complex financial fraud monitoring, translation engines, computer vision, reinforcement learning and complex ensemble models often need a large amount of distributed horsepower. Particularly when it comes to deep learning training on laptops or desktops or even in a datacenter with limited compute is simply not fast enough. It requires an array of chips and the ability to parallelize training across them.

The vast majority of the platforms profiled here in this report include some kind of training capability. Whether they are large clouds like SageMaker, Google Vertex or Azure Machine Learning, dedicated big-data and analytics platforms like DataRobot, Databricks or H2O, or mid-size startups like Valohai, Pachyderm, ClearML, Superb AI and Iguazio, all of them support training on a GPU. It should be noted, though, that many platforms execute a standard training script written by the user, while others provide additional capabilities to facilitate the process of single accelerator or multi-accelerator (distributed) training. Not all of them include distributed, parallelized training.

When we are talking about distributed training, we generally mean two capabilities:

- **Distributing training** across GPUs with little or no code alteration

- **Training multiple models** at the same time and comparing their results

Let's take a look at each in turn and see what they mean for your organization.

Distributed Training and Training Multiple Models

Distributed training is simply about taking a model that needs training and spreading out that training across GPUs to speed up the time it takes to complete the training. Platforms like Determined AI (now part of HPE) and open-source projects like Horovod that parallelize training across Tensorflow, Keras, Pytorch and MXNet deliver distributed training with limited or minimal code alteration, or by including specific scaling code into a team's workflow. Other platforms like TFX, the extended suite of open-source tools built around Tensorflow, include a distributed training engine with Trainer. Pytorch, the top competitor to Tensorflow for deep-learning work, includes Distributed Data Parallel (DDP) for paralleling work across GPUs on a single machine, while a more experimental distributed RPC framework allows for potential parallelizing across multiple machines. Cloud platforms like Azure Machine Learning leverage native Pytorch and Tensorflow parallelism to create their cloud-based trainers. Data parallelism is also possible across Spark clusters, but it is not ideal for training large language models or massive machine-learning models. BigDL is a library for building model parallelism on Spark, and it allows for support of Tensorflow, Pytorch, Keras and more, along with the running of compute frameworks like Ray on Spark, and the Orca library helps support that from a notebook. At the AIIA, we are also closely watching the emergence of new paradigms for training, such as the novel approach proposed by Juice Labs called GPU over IP, which promises better utilization and easier partitioning of GPUs at a distance, though the technologies are currently in the experimentation phase.

The major advantage of a system like Horovod is that it's more general purpose and supports multiple popular frameworks, compared to single-platform distributed trainers like Pytorch DDP. If your organization is only using Pytorch, then a single-purpose platform could serve your needs, but at the AIIA we tend to favor more general-purpose frameworks that support the flexibility to expand to multiple tools. In addition, if you need to support both TFX and Pytorch Distributed, you are supporting two different architectures and two different potential points of failure and slowdowns.

There are two main types of distributed training: data parallelism and model parallelism. When we talk about distributed training, we are almost always talking about data parallelism as it is the easiest to implement and sufficient for most use cases.

Data parallelism basically means that the data gets divided into partitions with the number of partitions equal to the total number of workers in the cluster. The model gets copied to each worker node, and the node operates on a subset of the data. Each node computes the errors and predictions independently and then communicates its changes to other nodes to update their version of the model before doing the next batch of data. Each node needs the capacity to fit the entire model as the model is updating across all nodes rather than being a smaller model that is later combined into a single larger model.

Model parallelism is done at the algorithm level and is dependent on how much of the algorithm is task independent from the other tasks. The model tasks are then split across nodes, and they all train on the same data. In practice, this is rarely done except in research or in custom algorithms designed by people with a high degree of distributed system skills. However, model parallelism may become necessary when it is difficult or impossible to fit a model into a single node. SageMaker supports a distributed model parallelism library for Python that supports Tensorflow and Pytorch and can parallelize training on the Amazon cloud.

Training multiple models is the second major distributed training task. It involves training lots of models that are independent at the same time, when a company has a high degree of model usage and deployment. This is basically a systems-administration and IT-level problem. There needs to be enough compute available and proper scheduling of jobs across clusters, usually with in-house Kubernetes or in the cloud.

The ability to train multiple models may also be one of the key features of an AutoML platform like DataRobot's AutoML or Dataiku's platform. This essentially involves trying a number of known successful approaches to a machine-learning problem, like the use of a classifier or computer vision, where multiple approaches have achieved state-of-the-art results at different times in the evolution of ML.

There are also platforms that focus on data infrastructure for training computer vision models. Activeloop provides an optimized format for unstructured data, so users can stream their datasets while training ML models in PyTorch and TensorFlow. Activeloop acts as a data lake for unstructured ML, and offers in-browser dataset visualization, querying, and version control. On top of those features, Activeloop integrates with experimentation and labeling tools to allow rapid iteration on computer vision datasets.

However, training multiple models may also be part of an experimentation orchestration pipeline. A team may want to try different model types to see which works best or to tweak and tune the hyperparameters to see which ones produce the best results. For instance, Determined AI's platform allows a team to execute complex hyperparameter searches and visualize the different versions of the models as they trained, after changing hyperparameters across the models, to determine which change is producing the best results. Determined AI lets a data scientist spin up hundreds of model variations at the same time, rapidly schedule them across GPUs and compare the results with visualizations. It also allows them to kill a job that is poorly performing to reclaim GPUs.

Supercomputers versus the Cloud

Beyond approaches to distributed training and reasons for distributed training, there are two major approaches to how to actually achieve it in practice:

- The cloud
- Supercomputers

The cloud is usually the first choice for organizations with light to medium distributed training needs. Essentially, the cloud is a classic lease versus buy equation. Just as you may lease if you only use a car for a set amount of time each day and for a set amount of years and buy that car if your usage needs are more constant, you should consider choosing the cloud for distributed training if you're usage patterns are unpredictable, which falls into the leasing side of the equation. The major advantage of the cloud is it deals with the complex IT undertaking of managing networking, compute and storage and knitting them all together seamlessly, though there are on-prem solutions that do this as well and semi-private clouds that are essentially an evolution of managed services in IT. Clouds also tend to have the latest versions of GPU or special-purpose units like TPUs or FPGAs.

When it comes to clouds, there are two kinds. General-purpose public clouds like Amazon Web Services, Google Cloud and Azure Cloud and machine-learning-specific clouds for distributed training like OVH Cloud or Grid AI. However, there are differences between ML-focused clouds that teams need to consider extremely carefully. While Grid AI is based on Pytorch Lightning and focuses on Pytorch exclusively, OVH is a public cloud competitor that built a subset of Kubernetes clusters for ML as a subset of their offerings.

The advantage of general-purpose clouds is that most organizations are already using them and familiar with their capabilities. In addition, many of the clouds have layered their own set of AI/ML tooling on top of those distributed training systems, but they are usually instances of open-source libraries like Pytorch DDP, which makes them less general purpose. Of course, if your team is running your own Kubernetes cluster on the cloud, then you can run any application you want and take advantage of GPU containers and deploy a framework like Horovod.

Clouds built for machine-learning tasks and training, like Grid AI, offer the advantage of having built-in visualizations and intelligent awareness of your workload; however they face stiff competition from general-purpose public clouds, and some competitors in this space have already failed. It's arguable that simply having a training cloud for AI will not be a supportable business model in and of itself, so the AIIA's general recommendation is to stick with the public cloud or to run your own instances on the public cloud for distributed training if you need distributed training. However, organizations that create their own hardware as a secondary business but also run public or private hosting clouds, such as Lambda Labs' GPU Cloud, are likely to prove more sustainable because their business model is not built solely on competing with the commoditization power of the public cloud.

However, there is one major alternative to the public cloud and that is supercomputing platforms like NVIDIA DGX. These are essentially boxes with a large number of top-of-the-line GPUs in a single box connected by a proprietary fast interconnect. The DGX A100 comes with 8 GPUs and over 5 petaFLOPS of performance, and they are able to be chained together in what NVIDIA calls a SuperPOD, which can include 20-140 DGX systems. Recently, similar products like the HPE Machine Learning Development System and Lambda Labs GPU clusters have entered the market, offering comparable hardware (though still running NVIDIA GPUs), and we expect more hardware stacks to follow.

Large hardware suppliers like Supermicro often have their own GPU heavy servers; however, they usually lack proprietary interconnects or clustering software, leaving it up to a skilled IT team to unite them into an AI/ML training cloud. Lambda Labs also builds credible alternatives to NVIDIA DGX machines, but they don't currently support the ability to be stacked together like NVIDIA SuperPODs.

If an organization is training lots of models and continually updating them, then buying supercomputing platforms in a box is often more affordable than running models 24/7 in the cloud. This is recommended for advanced organizations that have fully embraced cutting-edge machine learning in their organizations. However, it is also possible to use the cloud in a hybrid model with on-prem or managed private instances.

Current Trends and the Next Five Years

As noted in the introduction, training can be one of the most significant bottlenecks for machine-learning pipelines outside of labeling. However, not all training jobs are created equal. Many organizations are not yet training a large number of models or training models consistently enough to justify advanced training capabilities.

However, as more and more organizations scale to large model factories where hundreds or thousands of data scientists need GPUs or tensor ASICs to experiment more frequently or for teams working with more advanced AI/ML techniques, having a powerful training environment becomes essential.

The various public clouds offer a wide variety of hardware and skilled IT teams that can build on them by creating their own Kubernetes instances and leveraging advanced open-source training frameworks like Horovod, Pytorch Lightning or TFX. Teams with less skilled IT departments can turn to public cloud ML frameworks like Azure Machine Learning or Google Vertex to train models. The most advanced teams can leverage supercomputing in a box with platforms like Lambda Labs or NVIDIA DGX.

The Next Five Years

We do expect more companies to build large data science teams and to start leveraging more advanced AI/ML techniques. Both scenarios require a large number of GPUs and purpose-built tensor ASICs to support those efforts. A large team may not be using the most advanced techniques, but thousands of data scientists competing for a small cluster of GPUs will quickly become a bottleneck in any organization.

Equally, a team doing advanced work will have longer training times on existing clusters and crowd out other teams. If you combine both challenges in a single organization, lots of simultaneous models getting trained at the same time and complex models training for a longer time, you have a potential disaster unless careful preparations are not made ahead of time.

Advanced compute is not something that can easily be set up ad-hoc, even on the public cloud, and we recommend that teams growing their organization or moving into more advanced use cases plan sooner rather than later to expand their ability to train.

We also expect more foundational models to come from large, well-funded AI research organizations like OpenAI and DeepMind and for a wider swath of organizations to take advantage of these more well-rounded, general-purpose models. Stanford coined the term foundational model, and they have formed an organization called the Center for Research on Foundational Models (CRFM) to study them, in particular their societal consequences and use cases. According to the CRFM, “Foundation models (e.g., BERT, GPT-3, CLIP, Codex) are models trained on broad data at scale such that they can be adapted to a wide range of downstream tasks.”

Foundational models will impact how organizations that heavily leverage AI do their work. It is likely we will begin to see a shift to large, well-funded research institutions building foundational models that will flow down to enterprises. Some of them will exist only via API with strict controls by the organization, but others will be released as fully trained models that enterprises can fine tune with their own datasets.

Even with the use of foundational models, we expect many organizations will still need training capabilities to fine tune these models and to support the training of their own custom models as well. However, over a longer time horizon, we may see a decrease in the need for large-scale training factories or the rise of dedicated training factories to support most enterprise needs. In the short term, however, most AI/ML is still cutting-edge, and we do not expect the full range of solutions AI/ML offers to be encapsulated in foundational models within five years. We do expect those models to make a larger impact, though, and to become a part of many organizations’ AI/ML ensemble solutions.

DATA VERSIONING, LINEAGE AND METADATA

Companies and platforms covered in this section include:

Databricks, Pachyderm, Liquibase, TerminusDB, DoltHub, Weights & Biases, DVC, Valohai, Arrikto, LakeFS, ClearML, Iguazio, Comet ML.

Introduction

Data versioning and data lineage describe the ability of data engineers and data scientists to keep track of different versions of their models, code and data as they change over time, often at the same time. These abilities are essential to create reproducibility in data science pipelines and to address auditor requirements.

Data versioning and data lineage are another set of terms that often get tossed around interchangeably, but they mean completely different things.

Data versioning means keeping different snapshots of data.

Data lineage is how we keep track of what is in those snapshots and how the data changes over time. Lineage follows the entire journey of data as it changes over time.

We've seen versioning and lineage in traditional software coding with early systems like CVS and Subversion and today with Git. However, versioning and lineage of data is relatively new and has different requirements.

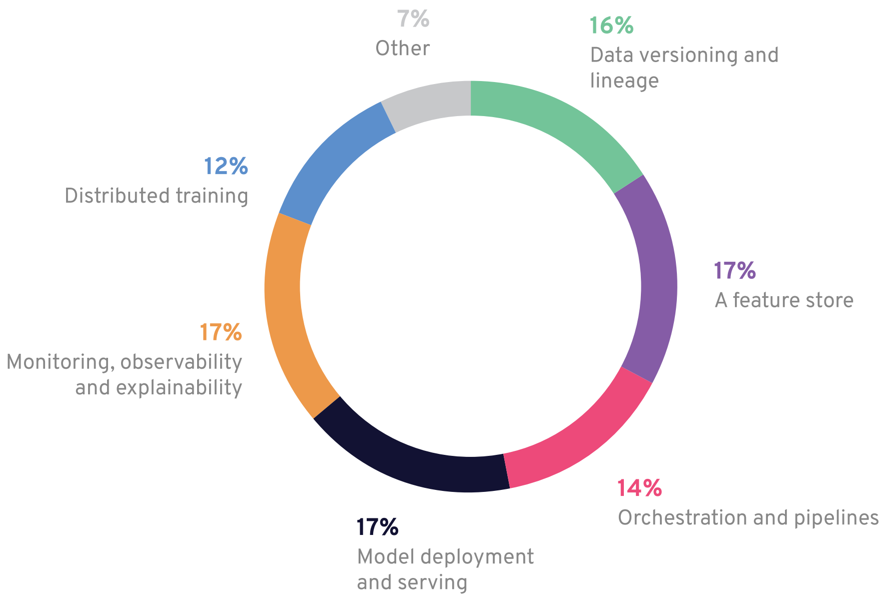
There are a number of approaches to data versioning. The AIIA divides these approaches into two types of versioning:

- Versioning of databases
- Versioning of files and/or object stores

Of the two, versioning of files has the longest history and the most development. Versioning of databases is a new phenomenon that largely mirrored the rise of machine learning, but it is catching on quickly.

In our survey, we found that data versioning and lineage together are one of the key aspects of the AI/ML infrastructure that teams expect to implement in the next six months to a year. They trailed only 1% behind the three aspects tied for the lead, which are feature stores, AI supervision and model deployment/serving frameworks.

What aspect of AI infrastructure are you looking to implement in the next six months to one year? Choose ALL that apply.



Many platforms have used the tagline "Git for data," and it's a good tagline. But what do we mean by Git for data? We know Git is version control for source code, but what does it mean to version data? As it turns out, the metaphor is not exactly apples to apples as versioning data has some major differences with versioning source code.

Let's take a look at each database versioning

Database Versioning

Database versioning is a relatively new entry to the field, but it serves the purpose of versioning and tracking the lineage of structured data.

When it comes to versioning databases, there are a number of approaches. At the time of this writing, none of the major database vendors, including Oracle, Snowflake, SQL Server, Redshift and Postgres, offer built-in data versioning, and the effect is achieved largely through plugins.

However, MongoDB can do some kinds of pseudo version control by using the very nature of the object database itself. For instance, they have a blog post on creating versions of documents by adding a field that keeps track of the version. However, this kind of version control is not true version control and generally assumes you are still mostly querying the latest version of the data and that you will not store too many versions.

When we discuss database version control, it's important to define the terms as there are several approaches.

One approach is versioning of the schemas. This is not precisely version control at the data level, which would include versioning of rows, tables and/or datums at the cellular level. A database schema is basically an abstract design that describes the organization of data and the relationships between tables in a database. It is essentially the blueprint or architecture of how data will be represented in a database. Schema version control is useful for database migrations but not for versioning of the data as it changes or updates.

The most prominent schema versioning system is Liquibase. Liquibase has broad support for most major databases on the market, including Snowflake, Oracle, Redshift, MongoDB and Cassandra. It is used primarily as a method to control updates to database schemas but not to keep versions of the data in the databases themselves.

In some cases, schema control can act as a sort of versioning of the data, in that the database can act as version control for slowly changing data. The database can mark data as active or inactive and show historical views of the data to users after changes, but it is not precisely database version control.

Planetscale is another scheme control system, similar to Liquibase, that has a broad customer base. The Planetscale team also wrote the Vitress clustering system, which enables horizontal scaling of MySql databases. In combination with their MySql clustering/scaling engine and their schema control system, they offer a SaaS-based service. Unlike Liquibase, their schema control does not apply to other databases, just MySql.

Full version control of the schema and the data is a relatively new concept in databases. There are several startups dedicated to making full database versioning available.

The first is TerminusDB. TerminusDB is an in-memory document NOSQL database. NOSQL databases come in three main types:

1. Key-value
2. Wide-column
3. Graph

Each type provides flexible schemas and scale well to handle large amounts of data and high loads. Specifically, TerminusDB is a graph database.

TerminusDB offers full schema and data versioning powers that let organizations create data intensive, immutable databases. It uses a custom query language called Web Object Query Language (WOQL), and it includes the ability to query JSON directly, similarly to MongoDB, which delivers a more document-database-style interface.

A graph database stores nodes and relationships instead of tables or documents. A graph database sees the connections between items as being as important as the items themselves. Graph databases address the challenges of many-to-many relationships, such as payment networks, social networks or highway and backroad networks. They allow developers to focus on the relationships between the data. Graph databases may serve teams well that are working primarily with relationships and using graph neural networks in production.

DoltHub is an SQL database with version and schema control. DoltHub uses the Git command line and associated operations on table rows instead of files. Modifications are made with SQL, and when someone finishes changes, they make a commit. This enables DoltHub to produce cell-wise diffs and merges. It offers merges and branches on the data itself. Finally, it acts as a drop in MySql replacement, but it is not backed by MySql but by the team's own database backend.

The most advanced of the database versioning systems is the Lakehouse architecture from Databricks, which is based on the open-source Delta Lake, an Apache project. Data is stored in database-style files called Parquet files, which are open-source, column-oriented data storage units. The files have a schema, and Spark supports ACID compliance with these files. They are excellent for storing structured and semi-structured data, but they are not ideal for unstructured data. They can store unstructured data, but it comes with tradeoffs, such as write speed and requiring large files like videos to be broken up into multiple pieces. They also don't have pointers between changed bits. Instead, Databricks stores versions as new Parquet files for simplicity but at a cost to storage space.

Despite some confusion created by its marketing name, the Delta Lake is a data warehouse architecture that excels at structured and semi-structured data. Other data warehouses include Snowflake and Amazon Redshift. Like Delta Lake, Snowflake relies on Parquet files to store primarily structured and semi-structured data.

Databricks has a relatively narrow vision of versioning. Versions are made only during experiment tracking, and by default they are kept for seven days, after which the older versions of the file are marked as tombstoned and deleted after thirty days, unless database administrators change the defaults. This means that data versions are not long lived on the Databricks platform and that the company largely sees versioning as limited only to the experimentation phase. If the goal of an organization is to keep long term, immutable versions of data, that is not currently possible with the DeltaLake architecture, which makes it ineffective for auditing or compliance or long-term reproducibility.

Short-term reproducibility is highly useful for teams doing rapid experimentation. However, when it comes to reproducing the exact conditions that created a model many months later, it becomes a problem. For instance, long-term reproducibility can quickly become a factor, as when a court or regulator declares a model illegal because it used sensitive characteristics or data that was not allowed to be combined with other data. In that case, short-lived immutability is not effective and the organization would need to turn to backups or to a different system or attempt to recreate the model conditions from scratch after deleting the offending data.

File and Object Store Data Versioning

File and object store versioning has a much more established pedigree and a basis in past computer science solutions like copy-on-write file systems.

At its most basic, we can do versioning by making copies of the data each time there's a new version we need to keep track of in our machine-learning pipelines.

In the example below, we have two files in a directory, cat.jpg and dog.jpg, and we call this version 1 or V1. The images are all 512x512 at 300DPI resolution.

```
images
|-- cat.jpg (1MB)     # 512x512, 300 DPI - Added in `V0`
|-- dog.jpg (2MB)     # 512x512, 300 DPI - Added in `V0`
```

We add a third file to the data set: bird.jpg (1MB)

```
images
|-- cat.jpg (1MB) # 512x512, 300 DPI - Added in `V0`
|-- dog.jpg (2MB) # 512x512, 300 DPI - Added in `V0`
|-- bird.jpg (1MB) # 512x512, 300 DPI - Added in `V1`
```

This creates a new version of the dataset, V2.

Now we alter the first file, cat.jpg, to make it smaller and reduce the resolution to 256x256 at 72 DPI. That gives us a third snapshot of our data, V3.

```
images
|-- cat.jpg (1MB) # 512x512, 300 DPI - Changed in `V3`
|-- dog.jpg (2MB) # 512x512, 300 DPI - Added in `V0`
|-- bird.jpg (1MB) # 512x512, 300 DPI - Added in `V1`
```

An intelligent data versioning system only stores changed data and uses pointers to refer to early versions, so it doesn't have to keep a complete copy of the directory for versions 1, 2 and 3. In other words, we don't have three directories with cat.jpg, dog.jpg and bird.jpg.

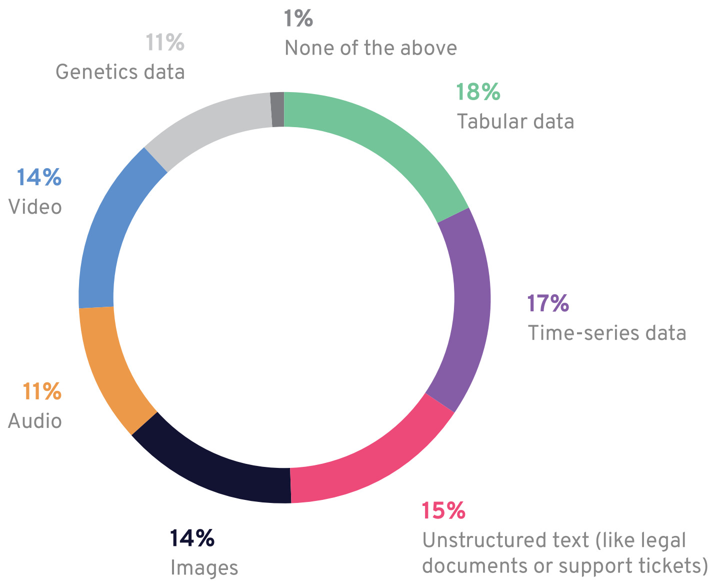
V1's directory contains the files cat.jpg and dog.jpg. V2 contains only the new file bird.jpg. V3 contains only the new 256x256, 72 DPI version of cat.jpg. The system then uses pointers to create the illusion that each dataset contains all three files.

Most often these files are stored in an object store or a file system overlaying an object store, such as MinIO, Amazon S3, Google Cloud Storage or Azure Blob Storage.

Object stores can store semi-structured or unstructured data. In particular, object stores excel at handling unstructured data like audio, video, images and unstructured text like Wikipedia articles and tweets, as well as semi-structured data like JSON files. It can also handle things like CSVs, which are not structured data but which can be converted into structured data. With decades of research behind large transactional and relational databases, structured data is still most often stored in transactional databases like Postgess, Oracle and Mysql.

While many early data science applications focused exclusively on structured data because it was better understood and easier to deal with, we're seeing an increasing number of teams make use of all kinds of unstructured data in their latest applications.

What kinds of data do you use to power your AI/ML applications?



There are advantages and disadvantages to each style of storage, and there are natural tradeoffs with each format. An object store will store a CSV file, which does not have the defined schema of structured data, but which can be imported into a structured data store or treated as structured. However, it won't have the same retrieval and write speed as a traditional database optimized for transactions, and it won't be able to enforce schemas or ACID (atomicity, consistency, isolation, durability) style writes to those rows and columns. For structured data workloads, a database still reigns supreme, and it also works well for semi-structured data, but truly unstructured files like a large video face significant overhead on reads and writes to databases or database-style files like Parquet files.

A number of companies have implemented data versioning as described above, including ClearML, Comet.ML, Weights & Biases, Databricks and Valohai.

For instance, ClearML, with its clearml-data system, allows for connections to various object stores, such as S3 or Google Cloud Storage, and has a series of commands to change, import, delete and update data to help prepare a dataset. However, it also includes a "finalize" dataset command that writes an immutable copy and allows no further changes in order to maintain long-term lineage, a sharp contrast to something like the Lakehouse short-lived lineage.

This type of version control is simple, clean and elegant, and it works well for smaller to mid-size projects with manageable data sizes, but it can prove troublesome for larger datasets, especially if the individual file sizes are large. If a team is working with 300 MB satellite images or 1 GB videos and making a full copy each time, there is a small change to each of the files that can grow storage costs fast.

Dealing with larger data sets and controlling space requires more advanced feature sets, such as deduplication. Deduplicated file systems are similar to copy-on-write (COW) file systems like Sun's venerable ZFS file system and Linux's Btrfs. Only changed data is copied when snapshots or shadow copies of the data are made. That results in significant performance storage space saving gains.

Let's take our original example and include bigger files to demonstrate.

```
videos
|-- cat.mp4 (200MB)     # Added in `V0`
|-- dog.mp4 (400MB)    # Added in `V0`
```

We make a simple change to one scene in the cat.mp4 video, altering just a few seconds of the 10 minute video. A simple copy based data versioning system will make an entire copy of the 200MB video for the new version.

```
videos
|-- cat.mp4 (201MB)     # Changed in `V1`
```

However, a COW-style, deduplicated file system will copy only the changed bits and use pointers to refer to the original parts of the file.

```
videos
|-- cat.mp4 (1MB)     # COW style, deduplicated file system - Changed in `V1`
```

Note the size differences between the two. In the deduplicated system, only 1 Mb is stored in the changed dataset, whereas in the directory copy system, the entire file is copied to the new directory.

AI/ML platforms that include COW-style, deduplicated versions of files include Arrikto, Pachyderm and LakeFS. Iguazio partners with PureStorage and other partners to deliver deduplication features.

One more feature is common in advanced AI/ML data versioning, and that is immutability. Immutability means that older versions of datasets cannot be changed or altered in any way. Immutability is essential to compliance and large-scale data versioning because without it a team can't ensure that the dataset has not changed out from under them.

Without immutability, it's possible for a metadata store or lineage tracking system to refer to a state of the data that no longer exists, which destroys reproducibility in AI/ML workflows.

Imagine that you have a directory in an object store with 20,000 jpgs at 1024x768 at 300 DPI. Your team runs a number of different experiments on that data. Now an administrator mistakenly overwrites the original jpgs in that directory with 256x256 versions at 72 DPI. The resulting compression results in a loss of fidelity that drives down an object detection model's accuracy. Since the original data was not immutable, your team can't go back to the original version of the dataset to recreate the higher accuracy model and must restore the data from a backup, if a backup exists.

Some versioning systems, such as DVC, don't enforce immutability by default, leaving it up to users to ensure their datasets are not changed in any way. AI/ML platforms that include enforced immutability are Arrikto, ClearML and Pachyderm, and on the database side, TerminusDB, DoltHub, Iguazio and Databricks.

Finally, it's worth noting again that platforms differ in how long they keep versions of data. Delta Lake has a retention threshold of seven days by default, after which older versions of datasets are tombstoned and set to be deleted by the vacuum operation. Other systems like ClearML, Arrikto and Pachyderm keep immutable snapshots indefinitely unless a project is completely deleted.

The difference is largely a design decision and also includes factors like complexity and storage size. In the case of Delta Lake, data snapshots are considered a temporary side effect of experimentation on AI/ML and analytics pipelines. This is likely because it's assumed that a team will settle on a final dataset after completing experiments. There is no reason that these files could not be kept longer other than storage considerations or massive growth in relationships between all the versions.

Other platforms, like Arrikto and Pachyderm, assume that different versions of datasets are long lived and that a team may wish to go back to an earlier branch of the data at a later time for multiple reasons, such as compliance or to recreate a different version of an experiment that was performing better than a production version, in order to compare the differences.

Data Lineage

If it's data versioning that keeps different snapshots of the data, it's data lineage that tracks those changes over time. You can think of data lineage as the entire journey of a dataset from beginning to end. How did it start? How was it transformed over time? Where is it now?

As snapshots are created, lineage systems keep track of when and where those changes were made, as well as meta information about what those changes include, so that users can perform diffs between a current version and a previous version.

When snapshots are infrequent and manual, such as when systems administrators are snapshotting a file system on a virtual machine, there is not much need for lineage. Simply naming the file appropriately is effective. But with automatic snapshotting and frequent snapshotting, it becomes impossible to understand what is in those snapshots without lineage. Snapshots proliferate rapidly and understanding what is in those snapshots quickly becomes a major challenge. It is also about understanding the relationships as the data, code, model and experiments are changing at the same time. All of that information is kept in a good lineage tracking system.

Sometimes a data lineage system is referred to as a metadata store. This is essentially another name for a database that keeps track of relationships and versions in a complex ML pipeline of ingestion, experimentation, training and deployment. There are two kinds of metadata/lineage stores:

1. **Wide metadata stores:** Stores that keep track of metadata from multiple systems.
1. **Narrow metadata stores:** Stores that keep track of metadata only on their own platform.

There are advantages and disadvantages to each.

The primary advantage of a wide metadata store is that it can aggregate lineage and metadata across lots of platforms. Since teams are likely to have more than one orchestration engine or endpoint, having a system that understands experiment runs across those systems is very useful.

The primary disadvantage of wide metadata stores is the primary advantage of narrow metadata stores. Wide metadata stores don't understand the underlying compute and capabilities of a platform as well as a narrow system, and wide stores often can't enforce rules on those systems or enforce immutability of data or models. Instead, wide metadata stores rely on the underlying systems to enforce rules or immutability. If that immutability is broken, the metadata store may refer to a state that no longer exists, like a pointer to a missing object.

InfuseAI, Neptune AI and Domino Data Labs are examples of wide metadata stores. You can usually spot them in the marketplace because they specifically call themselves metadata stores or refer to themselves as a "single source of truth."

InfuseAI's PipeRider platform gets around some of the challenges of immutability with ArtiVC, which can create snapshots for external object stores, with support for Google Cloud Storage, Amazon S3 and Google Cloud Storage. Metadata stores can leverage external capabilities of supported platforms, but they face the challenge of uneven capabilities across those platforms, whereas narrow metadata stores like ClearML, Pachyderm and others can take into account the underlying capabilities of the engine.

The vast majority of the systems mentioned here have their own tracking system for data lineage. Oftentimes the lineage tracking system is modeled on Git. Git is the dominant code tracking system in AI/ML and in traditional programming today, and its design philosophy and scale have outflanked older code versioning systems like CVS and Subversion. It's even spawned movements like GitOps to automated DevOps-style control of infrastructure. In AI/ML, Git is used for spawning pipelines and tracking algorithm/model code. It's also likely to remain the dominant platform for the coding aspects of AI/ML.

The question arises, why not just use Git, since it's already tremendously successful as a global, distributed code tracking system? It may seem natural to extend Git to deal with storage. But Git suffers from a number of design patterns that don't easily port to data even while those design patterns are perfect for code.

For instance, Git's approach to versioning is maintaining replicated copies of all files across every developer's workstation or laptop and then synchronizing changes over the internet and dealing with conflicts as needed. While this works tremendously well for code, which is usually small sets of files, it simply doesn't scale for large files and datasets which can easily choke bandwidth and local storage requirements. Data has gravity and weight. An MLOps team simply can't have terabytes of data copying to every workstation, so by and large AI/ML data version control systems centralize data by default.

DVC is one of the few platforms to use Git rather than take inspiration from Git, and as such it synchronizes datasets to local machines across the internet. Because DVC uses a familiar tool that developers already understand, it makes DVC a good choice for small data science teams and prototypes and non-data intensive workloads, but for datasets of significant size, beyond 20 GB, synchronizing data across the net quickly becomes untenable from a bandwidth and local storage requirement perspective. DVC does allow for the management of external datasets, but it's considered a beta feature and is not recommended by the team.

For most other platforms, the definitive choice is to centralize the data, either in a proprietary file system overlaying an object store, such as with LakeFS, Arrikto or Pachyderm, or by pointing directly to external object stores like S3, such as with ClearML, Iguazio, InfuseAI's ArtiV, Databricks, Dataiku, H2O, Valohai, Pachyderm and Arrikto. Proprietary file systems hold the advantage in terms of deduplication and direct control of a unified snapshotting method across clouds, but platforms that point directly to object stores hold the advantage that data doesn't need to be imported or converted to the proprietary file system, thus allowing data to remain where it lives.

The primary choice facing a company when choosing a lineage system is whether they need a narrow or wide lineage system. The simple solution is that for systems that rely on only a single core orchestration and pipelining engine, they will likely not need an external lineage system. However, if they are integrating multiple external engines, then they may end up having both types of lineage systems in their arsenal—the lineage trackers on the various platforms themselves and a wide lineage tracker for all of the systems.

Current Trends and the Next Five Years

While versioning and lineage is becoming more common with the rise of AI/ML, it's still a relatively new concept when it comes to data. Web applications and enterprise back offices have rarely needed snapshotting except for short-lived rollback and backup. But as data scientists continually adjust, clean, augment and change datasets, more consistent versioning and lineage becomes essential. Versioning and lineage are often challenges that teams discover late in their data science journey. Other challenges like serving and supervision appear sooner on their radar.

But once a team discovers they can't easily recreate an old model that's starting to fail in production or they have to replace a model due to an audit or because they found some edge cases that seriously affect the model, then the challenges of versioning and lineage become dramatically apparent. At that point, retroactively adding versioning and lineage is impossible. The damage is done. The AIIA encourages teams to make data versioning and lineage an earlier consideration in their platform journey.

When evaluating versioning, lineage and metadata, we suggest teams closely consider:

- Whether they need a narrow or wide metadata store or both
- Whether storage size is a consideration
    - If so consider an engine that supports deduplication
- Whether a system supports immutability
    - Without immutability a system can easily get into a place where it points to a state that is no longer true
- Whether it is possible to centralize all data or whether it must remain where it currently lives
    - If data must remain where it is currently, consider a platform with strong connectors to external data
- Take a look at the tradeoffs for centralizing data and leaving it where it is currently
- Look closely at long term lineage and snapshots versus short lived lineage and snapshots
- Consider that audit requirements are likely to increase in the coming decade

The Next Five Years

The holy grail of data for AI/ML is a unified storage engine for structured, semi-structured and unstructured data. The platform doesn't fully exist yet because of tradeoffs in storing each kind of data. As such, we're likely to see a continuation of multiple backends for some time with structured data continuing to live in SQL and backends, and semi-structured and unstructured data living in object stores or NOSQL databases.

However, over the next decade, we may see the emergence of a truly unified file storage backend for AI/ML, perhaps one that stores structured data in an SQL or NOSQL distributed database and unstructured data in an immutable, COW-style file system with pointers in that database that track its metadata and version.

We are also likely to see wide metadata stores that have better understanding of the underlying capabilities of external engines, but we don't expect those wide metadata engines to match the full capabilities of narrow metadata stores. It is likely that companies will continue to rely on both. Perhaps we will also see the rise of a unified standard for how to store and share lineage information that makes it easier for systems to publish and share data in a clear and consistent fashion across systems.

As noted earlier, perhaps the biggest change we expect in the next five years is a large increase in regulatory pressure from lawmakers targeting algorithms and how they are used. Algorithms, even though they are nothing more than an automated set of steps to solve a problem, have become larger than life in the popular imagination and popular concerns always draw lawmakers. We expect legislation to run the gamut from well thought out to poorly thought out and vague. But both kinds of legislation will increase auditing requirements and make data versioning and lineage move from being a nice-to-have to absolutely essential.

Versioning and lineage will serve two purposes as legislation increases. The first is that reports exported from systems will help craft reports that prove or disprove something to auditors. Second, it will allow data science teams to go back and recreate a model with or without data, logic or other models they can no longer interact with to meet regulatory compliance.

All of that adds up to the simple fact that data science teams should start thinking of versioning and lineage now rather than later to get themselves ahead of the curve.

LABELING

Companies and platforms covered in this section include:

Superb AI, Kili, Toloka, SuperAnnotate, Scale AI, Snorkel, Mindy Support, Amazon Mechanical Turk, SuperAnnotate, LabelStudio, Heartex

Introduction

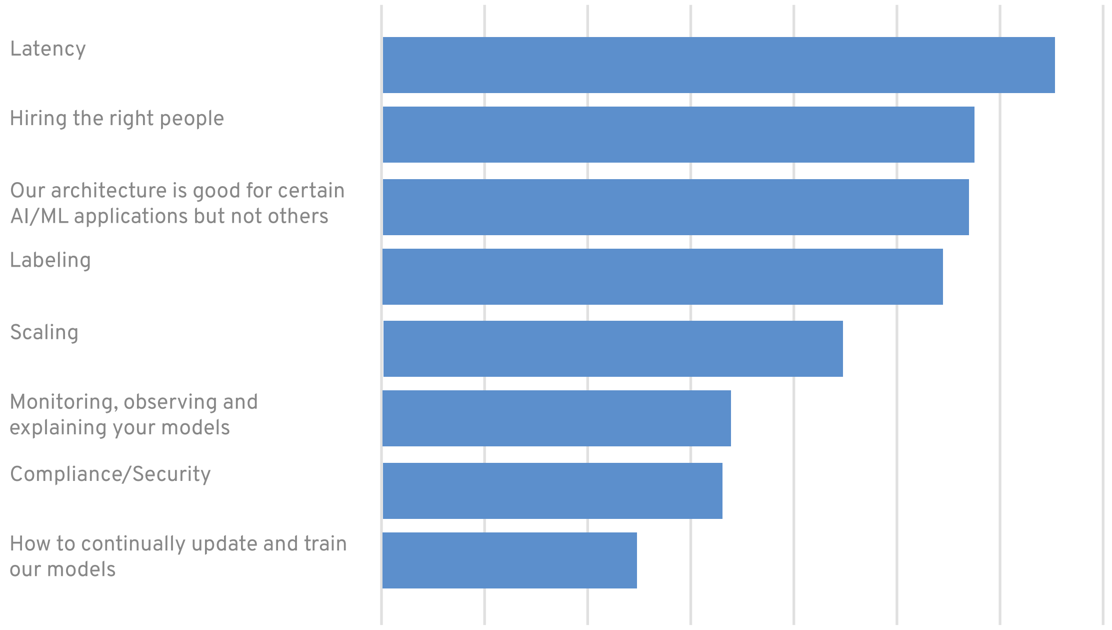
Data labeling is the process of adding tags and metadata to raw data like video, images, text, and audio. The annotated data codifies the kinds of knowledge and features we want AI/ML models to learn.

Data labeling is essential for all supervised learning, where an entire dataset is fully labeled, and a key ingredient of semi-supervised learning, which combines a smaller set of labeled data with algorithms designed to automate the labeling of the rest of the dataset programmatically. It's also particularly important to computer vision, one of the most advanced and developed areas of machine learning.

Beyond that, it's also one of the hottest areas of research in AI and one of the most well-funded sectors. In 2021 and 2022, we've seen label-focused platforms like Scale AI and Snorkel AI raise significant late-stage funding rounds and win some large contracts. The large contracts and customer wins indicate that many enterprises are working to label significant private stores of data that they've been unable to exploit prior to the rise of machine learning or have been unable to label due to the cost and complexity of the effort.

Labeling ranks as the fourth biggest challenge teams face when building their AI/ML infrastructure.

What are the most significant challenges you face when building your AI/ML infrastructure?



Data labeling is also one of the largest bottlenecks in AI/ML. That's because, historically, the only way to get good labels was through brute force or crowdsourcing because of the sheer number of labels needed to train effective models. Recent advances have helped augment manual data labeling by using machine learning itself to speed up the process with techniques like semi-supervised learning, which we discuss in more detail in the next section.

In addition to software in this area of AI/ML, many companies like Toloka, Kili, SuperAnnotate, and Scale provide a human labor force to create annotations along with domain experts to assist with how to label those datasets and consulting teams to apply machine learning and traditional programming methods to help programmatically speed up labeling. Some companies provide both human labeling teams and software, like Scale AI, whereas other companies, like Mindy Support, provide just a specialized labor force that can work with a variety of products.

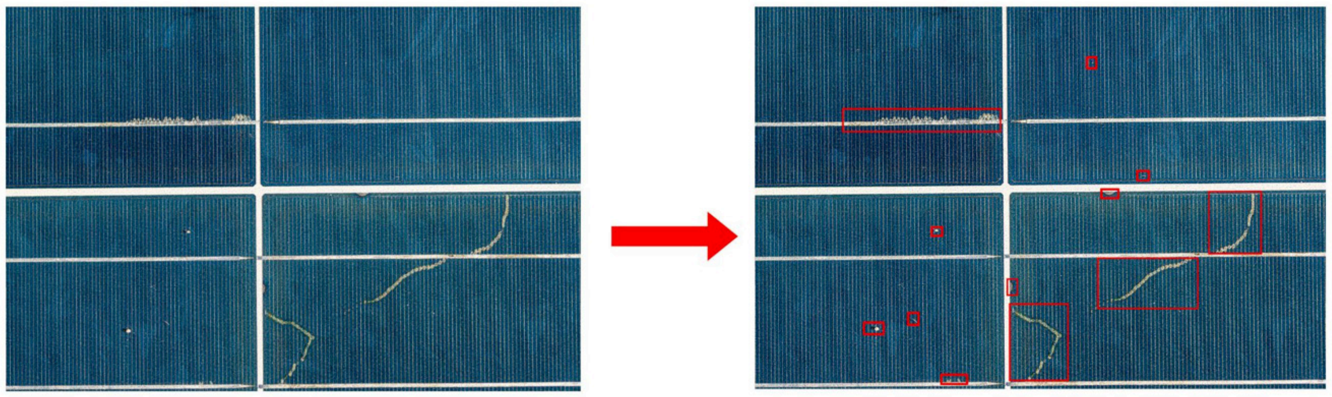
While many early efforts at data labeling relied on crowdsourcing or Amazon Mechanical Turk–style labor forces, the AIIA recommends that data science and data engineering teams work with trained data labelers as well as skilled teams, rather than ad-hoc, gig-economy teams, where the quality is often lower and labeling is decidedly less consistent. Alternatively, if your team has a truly massive project that requires some level of crowdsourcing, choose an organization that has software dedicated to crowdsourcing for AI labeling, like Toloka, rather than choosing a generic gig-data-platform approach, which will almost inevitably produce inferior results. Labeling tasks may also require significant domain expertise to be effective, such as with health-related data or legal data.

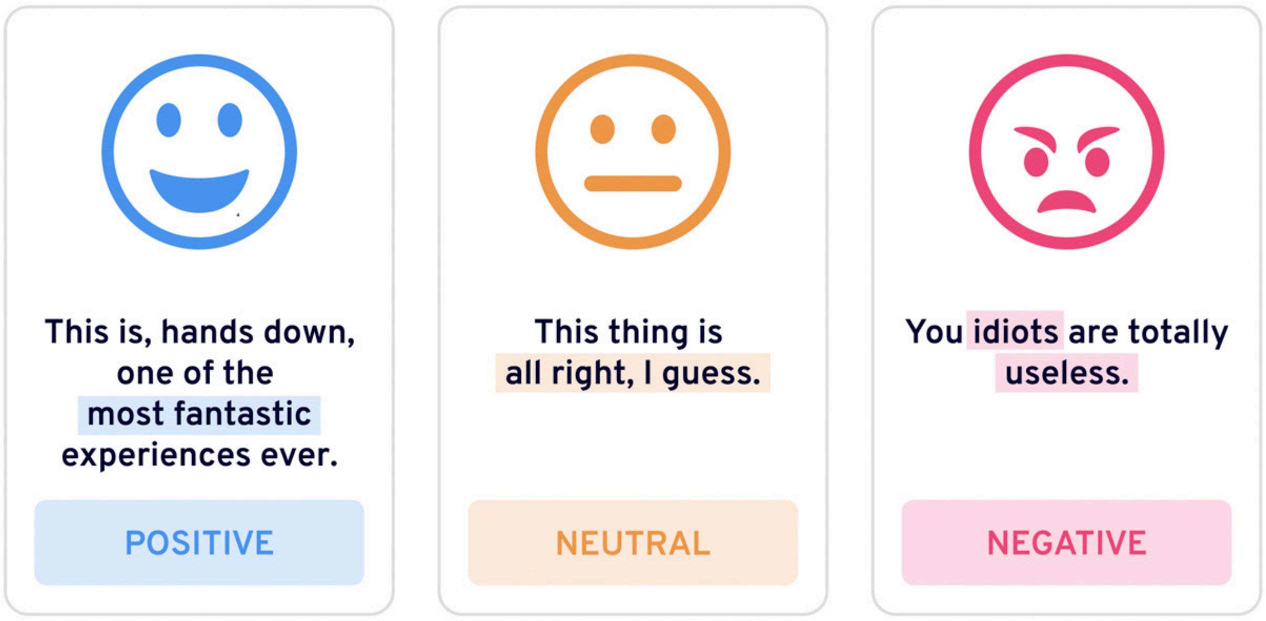
That said, there are examples where crowdsourcing is necessary either because of sheer scale or because of the embedded knowledge of the crowd, such as when Google Photos prompts you for help to label your own pictures. Nobody else in the world is better at recognizing you in a photo than you, so it makes perfect sense to tap your special expertise.

Understanding the Basic and Advanced Capabilities of Labeling Platforms

At their most basic, labeling platforms create a manual workflow that ingests data and allows a team of local or distributed human data labelers to label various kinds of data, such as text or images. With image data, labelers might put bounding boxes around specific objects and make notes as to what is in those bounding boxes.

When it comes to text, a labeler may highlight specific kinds of words, such as proper names or medical terms, or they may provide higher-level abstract information about the content of those words, such as the sentiment of the text or how one aspect of a legal contract relates to other clauses.




This is, hands down, one of the most fantastic experiences ever.
POSITIVE
This thing is all right, I guess.
NEUTRAL
You idiots are totally useless.
NEGATIVE

Open-source versions of these tools, such as Label Studio from the HeartEx team, are designed for small-scale data labeling or labeling by a group that doesn't require complex permissions or security. As is typical in open-source and enterprise application splits, the enterprise version of these platforms offers integration with existing role-based access control (RBAC) systems and permissions-based access to different projects or repositories, as well as visualizations, reporting and other enhancements.

All of the platforms surveyed here offer these basic labeling capabilities, including Superb AI, Kili, Toloka, SuperAnnotate, Scale, Snorkel and Heartex.

As we noted earlier, manual labeling is time consuming and expensive, even when partially automated. However, it is inescapable in many instances, and there are scenarios where it is one of the only ways to effectively work with a dataset. In cases where images contain lots of data with inconsistencies and domain-specific caveats and take a high degree of specialized knowledge to understand, manual labeling by trained human professionals is the best approach. For example, a computer vision model that looks to spot tumors in cancer patients should ideally be hand-labeled by trained radiologists.

Many teams want to keep their models entirely in-house, either for trade-secret reasons or because it allows them to leverage the advanced expertise of their own teams, who know the data inside and out. Tech giants like Tesla or Google will often use their own in-house teams because they are often working on advanced research projects, where talent and domain expertise is scarce.

Teams that have the highest demand for security will often opt for on-prem deployment or deployment to a cloud they control and will usually choose a platform that offers on-prem or local cloud deployment, such as Heartex or Snorkel.

These teams will also use an on-prem version when data gravity is a significant obstacle. Data gravity is the “weight” that a large dataset has when systems administrators or data engineers try to move it back and forth between different platforms or across the internet.

There are advantages to keeping data in-house, such as increased security and reducing the possibility of leaks. One advantage of doing this is that it keeps the labeling task with domain experts, and it also lets them adapt to rapidly changing datasets that would be difficult to export because of data gravity. Examples include things like self-driving cars, where new data from the field is constantly coming in that may offer additional edge cases for the ML models. But it’s also true with social media platforms, where new edge cases are coming in all the time, such as with the cat-and-mouse game played by people trying to get around advanced ML content filters with new and novel adversarial attacks.

The downside is that in-house resources are incredibly difficult to scale and using them takes up valuable resources that are best used doing more dynamic work. Labeling hundreds of thousands or millions of images is resource and people intensive, and most companies eventually turn to outside teams to support the task, or they completely outsource it (such as with a dedicated company like Mindy Support or with a company that has a large in-house team like SuperAnnotate). In addition, there are many tasks that don’t face significant data gravity and where security is not a top concern, and that allows teams to use SaaS-based platforms like Snorkel, SuperAnnotate, Scale and Superb AI.

Beyond manual labeling workflows, more advanced platforms allow for checkpoints, where managers can check the labelers’ work and adjust instructions to make the labels more consistent. People may interpret instructions very differently, which creates varying consistency in the quality of the labels, so the workflow must allow for checkpoints and the ability to incorporate client feedback into the process early and often.

They may also offer features like project management to track overall project progress, annotation statistics to see which labelers are generating the most accurate labels, reports to track team performance, and charts and dashboards to help visualize the performance.

Finally, the most advanced platforms look to help speed up the process by programmatically automating or heavily augmenting the process of generating labels. Each platform looks to differentiate itself in the market by offering the most ways to speed up various manual labeling tasks or complete labeling tasks entirely programmatically. Almost all of these advances involve machine learning itself, baked into the platform backend. This is an example of “turtles all the way down,” with ML being used to speed up the ability to create more advanced ML models. As such, the more novel ML solutions a labeling platform can invent to solve advanced use cases, the more useful a platform becomes.

Because of this, the labeling space is less crowded than other aspects of the field because it requires significant investment in R&D and smart, creative, highly skilled engineers.

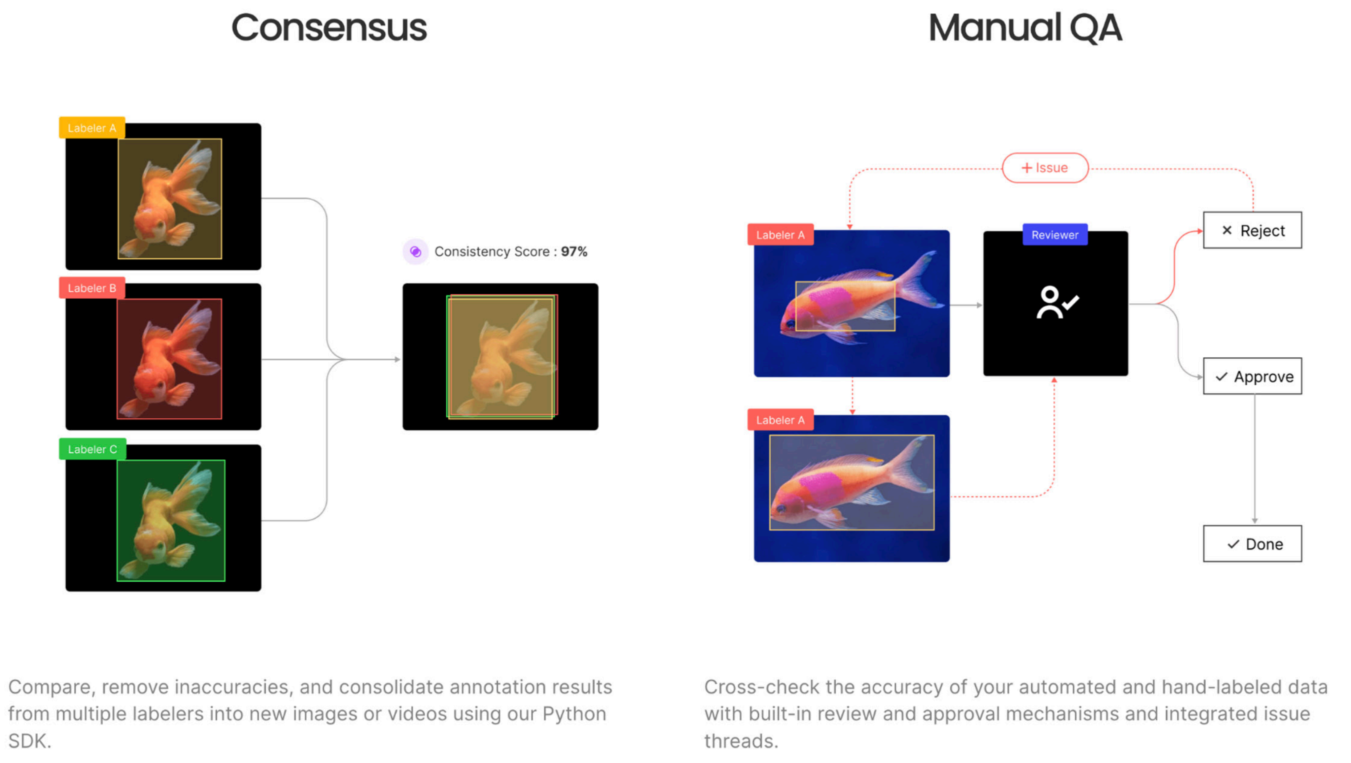
Advanced Capabilities

Advanced capabilities fall into two broad categories:

- Capabilities that use traditional programming
- Capabilities that leverage machine learning itself to speed up the process

The first category leverages traditional, hand-coded logic to help solve problems associated with the error-prone task of hand labeling a large dataset.

An example is SuperbAI's system checks for consistency in computer-vision labeling tasks, such as checking that bounding boxes are the same across different labelers. The system then allows a manager to reject or approve certain labels or send them back to be redone.


Consensus
Manual QA
Labeler A
Labeler B
Labeler C
Consistency Score : 97%
+ Issue
Labeler A
Reviewer
Labeler A
× Reject
✓ Approve
✓ Done
Compare, remove inaccuracies, and consolidate annotation results from multiple labelers into new images or videos using our Python SDK.
Cross-check the accuracy of your automated and hand-labeled data with built-in review and approval mechanisms and integrated issue threads.

(Source: Used with permission from Superb AI)

Machine-learning approaches generally fall into the category of semi-supervised-learning approaches, where a small subset of data is leveraged to train a model, rather than a completely labeled dataset. In this approach, semi-supervised learning is turned back on the labeling process itself. Human labelers label a portion of the dataset, and the label platform's model learns from that labeled portion and then attempts to automate labeling the rest of the dataset.

Kili provides a fairly unique approach to semi-supervised learning with their models learning in the background as a team does labeling. It then begins to make predictions as the human labeler is working, which then prompts the worker with suggestions to help them speed up.

Snorkel has a specific tool for named entity recognition (NER), which is a way for companies to tell the system how to recognize specific terms like medical terms, companies names, proper names, in-house jargon for tasks like risk classification or marketing, and legal terms. Since no public datasets exist with this information, it's up to an organization to supply the information and then use semi-supervised learning to annotate documents.

While this approach can be very effective, it still requires human-in-the-loop review and spot checking, as well as error correcting, especially for edge cases the model simply misses. There are many publicly available labeled datasets for sentiment analysis because human emotions are generic across text, whereas domain-specific knowledge like a company's personal business relationships simply won't be available. Weak supervision models will vary dramatically in accuracy and coverage. They often suffer from severe imbalance problems, where the model learns one feature very well and finds all of those instances but misses another feature entirely or spots that feature only sporadically. In other words, these techniques can prove to be more trouble than they are worth if they create a very noisy set of labels with lots of edge cases that require as much human intervention as hand labeling the data from scratch in the first place, but in general most teams will benefit from semi-supervised approaches.

Other approaches apply transfer learning. For example, Scale, Snorkel and Superb AI all provide a range of pre-trained models that can be tweaked to learn from user-uploaded datasets. A model previously trained on recognizing certain kinds of objects can help automate the creation of bounding boxes. This is similar to the magic eraser tool found in Google Photos, which automatically highlights people in the background of a picture and attempts to remove them to make the picture more picture-perfect.

Companies often use their own and open-source pre-trained models in an ensemble to speed up labeling tasks. The effectiveness of these solutions varies based on the dataset used. Common use cases like object recognition of people, roads, buildings and the like are likely to see a significant speedup, whereas use cases where datasets are harder to come by or unique, such as detailed medical imaging or small manufacturing datasets, will get less uplift from transfer learning, although datasets with similar low-level features may see interesting results as well.

The most advanced use cases are not pre-built but involve custom machine-learning consulting. Many of the larger firms, like Snorkel and Scale, have highly skilled data science teams that can work with highly technical clients to custom-build models for their use case, as well as custom heuristics and logic.

For example, Snorkel consulted with Google to help them build a rapid labeler for a classifier that replaced six months' worth of hand-labeled data in thirty minutes, but the work involved Google's programmers, Snorkel's data science team, domain experts inside Google and more to make the project work. The challenge of this kind of work is that it often takes a large effort to solve a single use case (classifiers) that can be transferred to other classifier use cases but perhaps not to other labeling needs.

To make it work, they had to develop an entirely novel approach that applied “a synthetic rebalancing and augmentation technique [that could] handle a high class imbalance that is very common in practice. Such an imbalance makes hand-labeling training data prohibitively expensive and causes problems for existing weak supervision approaches.” Their novel ensembling technique delivered a higher level of precision and proved valuable for Google. However, the primary challenge of this kind of approach is that not every company will have the resources or skills to work with an advanced team and, unless they are working with a lot of models of a specific type, that work may not transfer to their other labeling needs.

That said, companies that do have a need for lots of models and a lot of datasets that need more rapid labeling will continue to benefit from advanced consulting in this space.

Current Trends and the Next Five Years

Labeling remains one of the most consistent bottlenecks in data science. Of course, not all projects require labeling. All data have labels, but sometimes those labels are simple and auto-generated, such as with log data that indicates whether a customer clicked or didn’t click, or bought or didn’t buy. But many of the most advanced use cases like legal and medical document analysis, computer vision in many domains, self-driving cars, robotics, disease analysis and detection, drug discovery and more, require well-labeled datasets to work.

The basics of labeling software are relatively simple. It requires a workflow engine for labelers, a strong GUI, RBAC, manager overview and human-in-the-loop capabilities. But beyond that, the largest differentiators in this space come from machine learning itself, either from advanced machine-learning approaches built into the product or from consulting teams building custom novel approaches to a particular customer’s problem.

While small data science projects can easily use in-house labeling, we encourage teams working with significant datasets to work with external teams to speed up the process of labeling. It’s better not to burden employees who have the most domain knowledge with massive, repetitive tasks that are best left to outside teams.

When evaluating labeling platforms, we suggest teams consider the following suggestions and questions:

- Look closely at the workflow and UI.
- Do your datasets have unique or common features that may benefit from semi-supervised or transfer learning?
- Will you have the budget to build a custom labeled solution for your team?
- Will the resulting model you’re building have a significant effect on sales, cost reduction or the bottom line?
    - If so, then spending on consulting may justify the cost.
- Look to leverage your domain experts but limit their repetitive tasks.
- Consider your data gravity and how big your datasets are when working with SaaS versus on-prem solutions.

The Next Five Years

While this domain of machine learning has seen significant investment and advances in machine-learning techniques applied to the process of labeling itself already, we expect to see continued innovation here over the next five years and beyond.

More than anything, this space will benefit from multi-modal and more generalized AI approaches. Too often the solutions to a single client's use case are highly specific to their data, and the custom solutions don't generalize much beyond that customer's data. To truly see a breakthrough in labeling, we will need more generalizable techniques. We expect that portions of these custom solutions will start to show consistent patterns across many of the cases, which will, in turn, result in more general solutions for a wider subset of customer domains.

In the short and medium term, we expect labeling to continue to be one of the most bottlenecked parts of the data science pipeline. As more and more deep learning gets commoditized and becomes accessible outside of big research teams and small, innovative AI-driven companies, making its way into enterprises big and small, we expect more teams to need data labeling. As such, we expect the larger players to continue to gain mind and market share, but we also expect a surge of smaller players to catch on and grow quickly, especially if they are able to develop novel ML techniques that are highly accurate for labeling tasks. Those new ML techniques don't even need to be generalized across all domains. A breakthrough in the rapid labeling of medical data or manufacturing data could drive a smaller labeling platform to prominence and have it competing with the larger and more well-funded platforms fast.

Because of that, we simultaneously expect to see newer players in the ecosystem in the coming years and acquisitions of promising startups that capture a segment of a specific market.

FEATURE STORES

Companies and platforms covered in this section include:

Tecton, Feast, Molecula, ClearML, Iguazio, Amazon SageMaker, Google Vertex, Snowflake, Amazon Redshift, Kafka

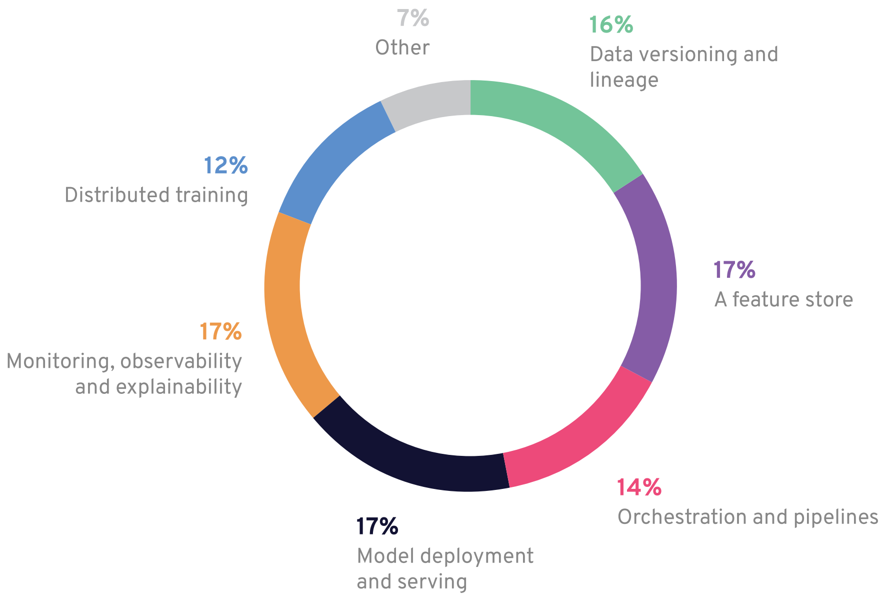
A features store is a data storage system for features of machine-learning models. Think of it as a central place for storing raw, curated, documented features that can be used across different models and data science teams. Essentially, it is feature management.

As a quick refresher, a feature is an input signal to a predictive model. The simplest example of a feature is a calculation of a person's age from the raw data of date of birth and the current date and time.

More advanced signals help us make more advanced predictions. For instance, if we're building a model that predicts when a pizza will be delivered, we need to pull from a number of different data points and do calculations on those data points. Raw data might include all the delivery times of every driver, and a feature might be the average time of delivery across those delivery times. Another example comes to us from credit card models looking to predict fraud. Useful features might be the size of a transaction and whether that transaction is happening in a different location than typical transactions.

Feature stores have generated a lot of interest over the last few years, seeming to appear in nearly every MLOps diagram almost overnight. Indeed, we discovered that feature stores are tied as the number one component of AI/Ml infrastructure that teams expect to implement in the next six months to a year.

What aspect of AI infrastructure are you looking to implement in the next six months to one year?



However, we found a large degree of confusion around why teams would actually need one and when they would need it. The short answer is that **large data science teams** working with **lots of models** that have **similar characteristics** are the right consumer of feature stores. Lastly, their use cases shoul primarily use **structured data** or **textual/ numerical data.**

Feature stores are almost exclusively structured data solutions. It is possible to store unstructured features in a data store, and platforms like [Iguazio's custom feature store]() allow for storage of unstructured features, however most feature stores do not support unstructured workloads (although, through exercises like data labeling, unstructured data can be converted into structured data that might be contained within a feature store). The reason is simple. Feature stores are built as a way for data scientists to browse or call features. As such, a data scientist is making a value judgment on the usefulness of those features and whether to include them in their own model by inspecting them with their own eyes and critical judgment. It's easy for a human to understand "average time of delivery" because it's written in plain language, but a string of numbers representing vectors in a computer vision model does not make sense and is not easily consumable.

There are four major components of most feature stores:

- Two databases
- A feature catalog
- Transformations
- Serving and supervision

The data warehouse is the foundation of the first database, and it includes large amounts of **offline** features for performing offline model scoring and creating training datasets. Typically the backend is a traditional data warehouse like Snowflake.

The second database is a database that offers online features at low latency. The key here is low latency, and feature stores like Feast, an open-source platform largely supported by Tecton, Feathr, an open source feature store created by LinkedIn, and Molecula's low latency featurebase platform offer sub-second latency. Low latency delivers the freshest version of features to models as fast as possible, in essence acting like a caching layer for features.

The next major component of a feature store is a feature catalog. It also allows for historical calls to older versions of features. This allows data scientists to visually browse features or call them programmatically via API. Data scientists register their features in a feature registry at the training stage. Those registrations include:

- A feature definition
- Metadata

A feature definition specifies the data transformation that will take place. That might be as complex as triggering a training pipeline via an orchestrator to refresh the feature, or it might be a simple as an SQL or NOSQL query. The metadata is data about the feature, such as who created it, the version number and whether it is a stable production feature or experimental. Metadata may also include documentation about the feature.

We've already touched on the third major component: transformations. A feature is not present in the raw data. It needs to be calculated through a pipeline or via a query and/or a set of data transformation steps.

There are two kinds of transformations:

- Batch
- Streaming

Batch transformations might happen right inside another data warehouse, like Snowflake or Amazon Redshift, via a query. Or the transformation may take place in an orchestrator or pipelining engine like Apache Spark, Databricks's Spark SaaS or another pipelining engine. Feast, Feathr, Tecton and Molecula all support Databricks and Snowflake as data warehouse backends. Feathr also supports a range of additional backends, most notably object stores like

Azure Blob Store and AWS S3, which makes it potentially possible to support additional pipeline systems like Valohai, Pachyderm, ClearML, Kubeflow and other orchestrators that ingest and egress to object stores.

The platforms also support the ability to do streaming transformations. These are transformations usually done on a platform like Kafka. This allows for transformations that need more up-to-the-minute data like the average number of users on a website or the number of orders for hairdryers in the last half hour. Molecula, Feathr, Feast and Tecton all support Kafka as a backend for streaming transformations.

Serving and supervision are the last components of a feature store. This is how feature stores serve features, which is usually via an API from the low-latency database. Advanced commercial feature stores also take into account replication, clustering and high availability. Finally, feature stores must be included in traditional IT monitoring platforms to ensure that they are not down or suffering from a sudden surge in latency.

Current Trends and the Next Five Years

We've seen a number of commercial feature stores come to market over the last few years. They are facing growing pressure from open-source solutions, and they're also facing competition from orchestrators with their own proprietary built-in feature stores, such as ClearML, Iguazio, Databricks, Amazon SageMaker and Google Vertex.

Closed platforms with tight integration, like Amazon SageMaker, have feature stores that integrate only with their pipeline, and the features saved on the platform end up siloed and inaccessible to other platforms, whereas a commercial platform or open-source platform allows access to features via API, which allows them to be consumed by outside platforms. If your primary orchestrator is a single platform, then a built-in feature store may serve your needs, but take caution that it doesn't trap those features in that platform alone. Look for a feature store that allows access via API to outside platforms.

Commercial feature stores are good for large teams that need guaranteed response times and dedicated support. SaaS offerings may help manage complexity. However, they may not support all the desired backends that an organization needs. Open-source solutions are good for smaller teams, teams that have excellent IT skills and teams that need to support a wide variety of backends or extend the platform to add their own backend.

The Next Five Years

AI/ML has very different requirements in academia and production. While academic AI research focuses on maximizing benchmark scores, regardless of size, production AI is about manageability and latency. In academic research, data is usually held fixed, but in production it's constantly updating and growing, and models need retraining to stay fresh. As we've seen, production-grade AI/ML has very different requirements. You need massive amounts of data that must be preprocessed in a variety of ways (cleaned, transformed, feature engineered, etc.), and all of this must happen within milliseconds for a prediction to be accurate.

In the next five years, it's likely that the ecosystem will start to converge on one or two open-source platforms with a vast amount of backend support and increased openness. We expect to see those open-source platforms begin to displace some of the proprietary, built-in platforms. It's likely that orchestrators will just swap built-in versions for well-regarded and well-maintained open-source solutions.

We expect most large tech organizations to stop building their own feature stores as they will mostly just be reinventing the wheel. Instead, they will likely purchase a feature store, deploy an open-source one or consume a commodity feature store in their orchestrator of choice.

We also expect to see public clouds offering commercial support to those leading projects or individual companies springing up to offer support for and configuration of those platforms, just as MongoDB supports scaled versions of its open-source offering to help organizations manage complexity.

As mentioned above, feature stores typically include an offline as well as an online store, which can cause discrepancies between the data used in training versus the data used in production. In the future, we predict there might be an opportunity to unify offline and online stores into one low-latency feature storage solution that enables training and productionizing of models on the same feature sets from the same feature storage layer. We're starting to see this with the creation of feature-specific binarized databases that are able to meet the challenges of enabling low-latency querying directly on raw data without the need for pre-aggregation or preprocessing. Essentially, there is a potential opportunity for a highly efficient compute layer, built around features, to emerge between data storage and the application layer, removing barriers that currently exist when it comes to productionizing models in near real-time.

We also may see the rise of feature stores that support unstructured data, which would be connected to the rise of an external framework to automatically consume them, rather than having a data scientist as an intermediary interpreting the features they want to consume. For example, we can foresee the possibility that features of a powerful object detection model could be consumed by a smaller model that is fine tuned to a particular task, in essence creating a computer vision micro-model chain. The capacity for a low-latency database to store these kinds of features is fairly trivial and possible now. The real development would be a programmatic way to consume them and that perhaps might develop into an ML library category of its very own.

Lastly, at this point, it is challenging to predict whether feature stores will become a commodity of the AI/ML lifecycle over the next five years, with most platforms including them by default, or if a dedicated feature store will remain a scalable business model on its own. Largely that will depend on whether those commercial feature stores can aggregate transformations from many different backends, manage complexity that organizations would rather avoid and offer low latency that's simply much more high performance than open-source or built-in alternatives.

SYNTHETIC DATA

Companies and platforms covered in this section include:

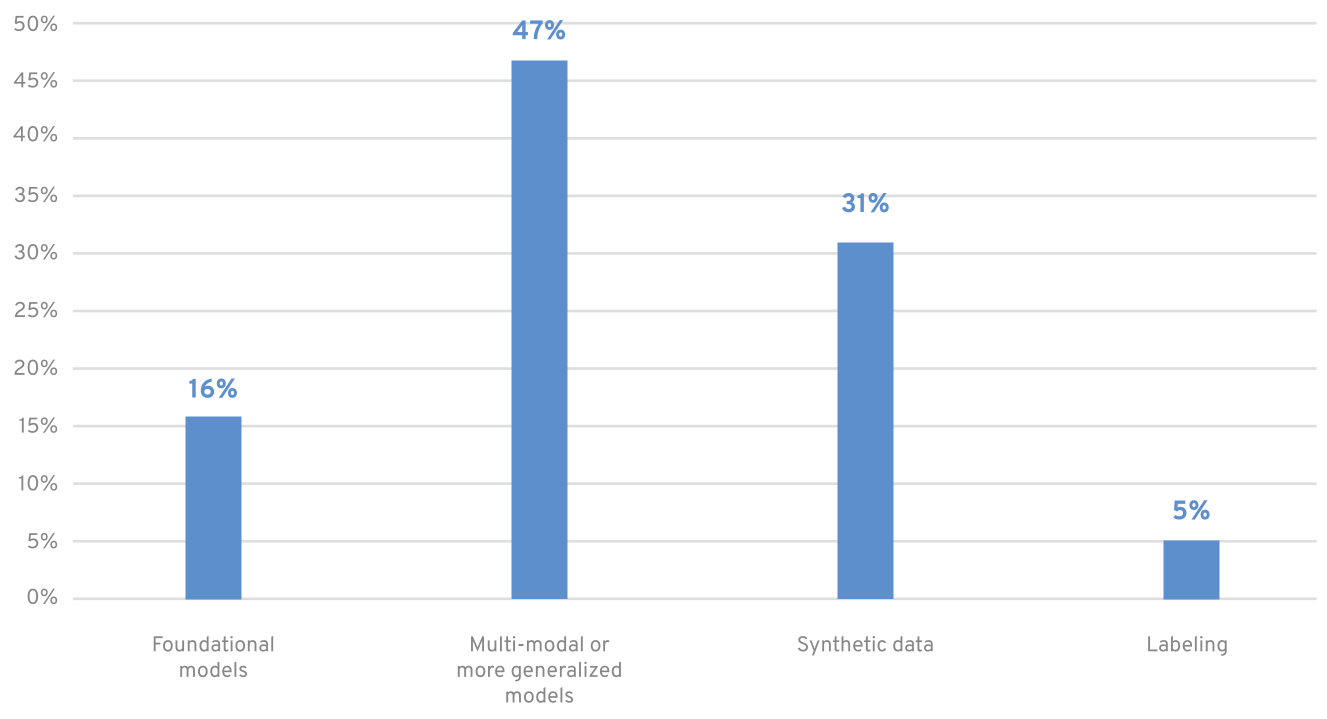
Gretel, YData, Mostly AI, Statice, Hazy, Synthesized, Diveplane, Mirry.ai, Replica Analytics, Sogeti, Syntho, Tonic, Anyverse, Bifrost, CVEDIA, Cognata, Coohom Cloud, Deep Vision Data, Neurolabs, Parallel domain, Rendered AI, Scale, Simerse, Synthesis AI, Synthetic Data Vault, Twinify, EdgeCase, Unity engine, Syntegra, MDClone, Facteus, Pasteur iSi Simulation Science

Synthetic data is artificial data that isn't gathered from the real world.

It's one of the most cutting-edge domains in the AI/ML ecosystem. It's also one of the newest entries to the AI/ML landscape, and most of the platforms here were created in the last two years, with no company older than five years.

But many companies we surveyed expect synthetic data to make a big impact on their model building in the next five years, trailing only multi-modal models, a.k.a. models that are able to do many things well.

What area of the AI/ML infrastructure do you think will have the biggest breakthrough in the next three to five years?



Just as labeling platforms often uses the power of machine learning itself to speed up the tedious task of labeling, synthetic data platforms use machine learning to bypass the gathering of data.

Synthetic data falls into three main categories:

- **Fully synthetic:** This means the data doesn't include any data gathered from the real world. It was generated entirely procedurally.
- **Partially synthetic:** Partially synthetic data replaces part of the data with generated data. For example, it may replace sensitive characteristics like names, social security numbers, addresses and the like.
- **Hybrid synthetic:** Hybrid synthetic data comes from both real and synthetic data. This is usually for data augmentation, often of a smaller dataset that needs more examples to be useful for training a model. It works by learning the distribution of the data and generating more elements of the data to produce a synthetic and real-world data combination.

Synthetic data generation methods come in three main categories:

- **Dummy/mock data:** This type of synthetic data is commonly used for testing environments. Its primary characteristic is that it's generated in a random manner or based on certain rules. A good example of a way to generate this type of data is Faker.
- **Statistics & Population inference:** Population synthesis generates records of individuals/events that behave closer to reality than dummy or mock data, but this kind of synthetic data is still driven by rules and wider population inference. The goal of this kind of approach is to generate individual records with associated attributes that closely resemble a population that leverages real aggregated information. It's based on microsimulation models that follow human-derived rules. We see this type of data leveraged for pandemic studies and the development of biotech and pharmaceuticals, but it can also be used for risk simulation in areas such as financial services.
  A good example of this is agent-based modeling.
- **Data-driven or ML/DL based:** With synthetic data generated through a data-driven approach, any behavior/rule inferred was learned from the patterns in an existing dataset. It is essentially a look-a-like data set that closely mirrors the properties of the real one, not only in terms of its statistics but also in what concerns the multivariate relations, either linear or nonlinear.

After building the synthetic dataset by leveraging the methods detailed above, we can end up with fully or partially synthetic data.

Hybrid Synthetic: Hybrid synthetic data comes from both real and synthetic data. This is usually used for data augmentation, often of a smaller data set that needs more examples to be useful for training a model. It works by learning the distribution of the data and generating more elements of the data to produce a combination of synthetic and real-world data. Each of these approaches helps to solve a number of real-world challenges, most notably:

- The difficulty of gathering real world data
- The challenge of finding enough edge cases
- Data augmentation

- Privacy and security

The first challenge is that it might be hard to collect or acquire the data a company needs to build an accurate model. It's not just the labeling of data that's time consuming. The very act of gathering data can be incredibly time consuming. Whether that's sending people out to take lots of pictures or to film video or simply waiting for data to come in from sensors in the field, it takes a lot of people hours. Companies might not have the resources to purchase a commercial dataset or to go out and create that dataset themselves. The advantage of synthetic data is that it can be generated rapidly and it can even be auto-labeled as it is generated, bypassing the two slowest parts of the AI/ML lifecycle.

The second potential problem is that it might also be challenging to acquire data because of the rarity of events. If we have a dataset of manufactured widgets from our factory, we may have a tremendous number of pictures of perfectly good widgets but not enough examples of broken widgets or widgets with manufacturing defects. Gathering those images would involve breaking lots of widgets and photographing them or waiting for more examples of broken widgets to come in.

Rarity is shorthand for **edge cases**. One of the reasons big-tech companies like Google or Tesla collect massive datasets is because they hold the potential of capturing the largest number of edge cases. For instance, the Waymo team building self-driving cars noted that they were "99% of the way to self-driving cars but the last 1% is the hardest." They were referring to rarely occurring events that humans are good at dealing with based on previously learned skills, but that machine-learning models need to understand through study because they lack adaptability when presented with the radically unfamiliar.

The third challenge is that it might simply be faster to generate a hybrid synthetic and real-world dataset. For example, when it comes to training drones, you may have a number of videos or images in sun and rain, but you need more examples of fog and snow. You can do that by generating the alternative conditions that augment the dataset rather than waiting to capture more rare conditions like fog. You can also take existing images of fog and increase the density of the fog to mimic more extreme conditions.

Lastly, synthetic data can minimize privacy concerns. Collecting and using sensitive data raises serious privacy concerns, and regulators have put increasing scrutiny on the practice with regulations such as the GDPR (the European Union's General Data Protection act) and the CCPA (California Consumer Privacy Act) restricting how organizations collect and use personal information. Fines can be imposed on companies that violate these restrictions.

The state of legislation and how it applies to real-world data and synthetic data is still evolving and will continue to evolve in the coming years. Legal teams are already looking at it and helping regulators come to terms with synthetic data. For instance, Telles, a law firm based in Portugal, analyzed the GDPR and notes that:

> Recital 26 of the GDPR states that the regulation does not apply to "information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable." Data synthesis would then benefit from the same reasoning as the case for not needing to comply with further obligations for anonymization under GDPR. However, depending on the synthesis process applied, the need to ensure initial data creation and testing must comply with applicable laws. Notably, the process of anonymization is itself a data processing activity that needs to comply with data protection laws.

In other words, it depends on how the original data was used, the method of generating that data and more. We expect every team working with sensitive data to need a strong legal team to understand all the regulations and how they apply to them and their model development.

Even when organizations attempt to anonymize real data, it can prove to be incredibly difficult. A research team studying attempts to remove sensitive characteristics from resumes found it was nearly impossible. Even when removing so much of the resume as to make it unusable for HR teams, machine-learning models were still able to predict gender and other characteristics 70% of the time. That's because even if sensitive or identifying characteristics are removed or obscured, other variables can act as proxies to that information.

Synthetic data can help with privacy in two main ways. If the data is fully synthetic, then it was never based on a real-world person or event. A partially synthetic dataset can avoid data leaks such as a model spitting out an API key in a code generator. It can also create a dataset where something like names and social security numbers are replaced so that if the data leaks in a breach, it's not real data that can be used by identity thieves. There are some caveats to that, in that the original dataset must be deleted or kept air-gapped or in a more highly secure area, or else the original data might leak.

Synthetic data generation leverages state-of-the-art concepts in machine learning, and researchers are always looking at the latest techniques coming out of research labs to see if they can be applied to synthetic data, but currently the techniques most often used are:

- Bayesian networks
- GANs (generative adversarial networks)
- VAEs (variable auto encoders)
- Flow based models
- Diffusion models

Bayesian networks use Bayesian inference, one of the most battle-tested statistical reasoning methods, to create a probabilistic graphical model. Bayesian networks look to model causation by finding dependency conditions and then representing that conditional dependence in a directed graph.

Generative adversarial networks were designed by researcher Ian Goodfellow and his colleagues in June 2014. With a GAN, two neural networks, a generator and a discriminator, contest each other in a game where the generator attempts to fool the discriminator until it gets good enough to consistently fool the discriminator. For instance, the discriminator might generate photorealistic faces until the discriminator can no longer tell the difference between the generated faces and the real pictures of people.

As you might have guessed, GANs have been used to generate realistic faces, fill in missing sections of a photo or text, unblur photos, upsample astronomical photos and generate realistic speech that's never been spoken.

However, although GANs can produce remarkable results, they can prove very challenging when it comes to training a stable model. But by its very nature, the training process is inherently unstable because it involves the dynamic training of two competing models at the same time.

An autoencoder is a type of neural network that learns how to efficiently encode unlabeled data via unsupervised learning. It validates the encoding by repeatedly attempting to regenerate the input from the encoding. It works by learning a representation of the data, typically by dimensionality reduction and learning to ignore noise. Variable auto encoders have the ability to generate variations on that data rather than just reconstruct the original data. That means if they learn a representation of a face, they can generate different faces rather than just the original.

Variable auto encoders are used in image and audio generation and in interpolating sentences to fill in missing words.

Flow-based models use a probability distribution that leverages normalizing flows that transform a simple distribution into a complex one. They've been applied to a wide variety of modeling, such as audio generation, molecular graph generation, point cloud modeling, and video and image generation.

Diffusion models, inspired by non-equilibrium thermodynamics, work by slowly corrupting the training data by progressively adding noise. Then the network learns to reverse the corruption by gradually denoising the data. The advantage of diffusion models is that they can be effective at learning and generating high-dimensional data, such as financial data or genomic data.

The DALLE-2 platform and Google's answer, Imagen, both take advantage of diffusion models and have helped advance this area.

A Survey of Platforms

Synthetic data providers tend to focus on one of two broad categories:

- Structured data
- Unstructured data

Just as there are differences between storing, processing and working with structured and unstructured data in pipelines, there are differences between how effective it is to generate synthetic data that's structured and unstructured.

At first glance, you might think structured data is further along, just as it is in the rest of the space. But currently, the use of synthetic data is the most developed in the image generation space. The reason for this is that it's easier for people to validate an image, they do so simply by looking at it. We have built-in instincts for whether something "looks right," but validating lots of structured textual data is much more challenging.

But if a sixty-character API string is off by a digit or includes a character that is invalid, could you recognize it at a glance or even with heuristics like a RegEx? Textual data takes more time to inspect and validate. Other unstructured data applications outside of image generation is still a work in progress, such as realistic document creation.

That said, largely the same ML techniques discussed earlier (e.g., GANs and VAEs) are used in both structured and unstructured data, with the difference being that structured data can also benefit from traditional hand-coded logic and heuristics.

We find that platforms tend to focus on either structured or unstructured data generation because of the challenges of creating both equally well in a broad range of domains and/or validating that the data generated is useful for training real-world models consistently in a specific domain.

We also find that many platforms are further subdivided by:

- Industry focus
- Category of synthetic data (e.g., partially synthetic data or fully synthetic data)
- Both of the above

We have found that many platforms have been designed to focus on specific industries, because many industries, such as healthcare or financial services, have their own unique challenges, and developing a platform that can simultaneously develop synthetic CAT scans as well as generating examples of financial fraud is still challenging.

For example, Facteus focuses on the financial industry, and its synthetic generation tool is specifically designed to generate partially synthetic datasets with fake personally identifying information using its mimic engine. YData focuses on structured (tabular and time series) synthetic data for financial firms, telecom, retail and more. One of their key focuses is on data quality activities, of the kind discussed earlier, such as finding over and underrepresented data that can destroy a model's chance of generalizing well into the real world. MDClone focuses exclusively on healthcare data and offering privacy and security that allows data scientists to explore original datasets and synthetic look-a-like datasets. Syntegra also focuses on healthcare and life sciences and includes on-prem deployment via container to eliminate the need to upload datasets that might be too sensitive. Companies like Pasteur iSi take a different approach, looking to simulate various environments with digital twins, such as a digital twin of the Earth.

The AIIA has noticed that fully synthetic generators, like the Unity engine for generating synthetic video and images from no gathered input data, are largely confined to the field of unstructured data, and skilled operators or consultants are required to create these datasets, just as skilled animators are required for animation. The vast majority of the companies surveyed by the AIIA on structured data offer platforms with the ability to generate synthetic data by first learning the features and distributions of existing datasets, which means most of the structured platforms do not solve the problem of generating new edge cases or novel use cases as seamlessly as the unstructured data platforms.

We've also noticed that the vast majority of use cases of the platforms in the survey do not include incredibly challenging capabilities like generating synthetic cancer data, where there would be large repercussions for failure, and instead focus on tabular look-a-likes of patient data for insurance policy creation rather than for drug discovery. That makes sense because it's easier to create a mirror dataset by modeling collected data than to generate novel, accurate, realistic variations from complex datasets like lung cancer or skin cancer datasets. However, a few companies, such as EdgeCase, are looking to tackle some of those more difficult scenarios, involving radiology data and other medical imaging data, and agricultural disease detection.

We have also noticed that the vast majority of the companies in this space are SaaS services because of the need for those companies to hire top-notch researchers and data scientists.

There are some open-source projects of varying quality, such as Twinify, which generates a privacy-preserving synthetic twin when given a dataset with sensitive characteristics. Ydata-synthetic and Gretel Synthetics generate tabular data. Synthea generates fake patient data. The Synthetic Data Vault project, which started out of MIT's Data to AI Lab, maintains a list of projects and tutorials and packages up some of those projects into a single project for easier access.

It's important to note that many of the open-source projects are still new and have not achieved the kind of third-party contributors that larger scale open-source projects have in recent years. We expect that to change as the synthetic-data market develops. We also expect more companies to craft open-source tools that get added into their larger portfolio of capabilities. However, at this point in the ecosystem's evolution, it's difficult to rely on pure open-source solutions, whereas companies can and do rely heavily on open-source solutions for other parts of the AI/ML stack, such as with Pytorch or Tensorflow.

Additional structured-data-focused solutions include:

- Gretel: Generates synthetic tabular data, performs privacy preserving transformations on sensitive data and offers NLP options for detecting PII in data.
- Diveplane: Their Gemini solution generates synthetic twins of data.
- Hazy: A financial synthetic-data generator that can be installed on site as a Docker container or used as an SaaS, with the data generated by analyzing existing datasets and producing a privacy-preserving alternative version of the data.
- Mirry.ai: Has a cloud platform that integrates with major databases like Snowflake, Oracle and Redshift and can be used for privacy preservation to correct data imbalances.
- Mostly AI: Their Mostly Generate platform creates look-alike tabular data for the financial, telecom and insurance

industries.

- Replica Analytics: Their Replica Synthesis platform generates synthetic look-a-like data. It includes an API, and development libraries in R and Python that integrate data synthesis into existing pipelines.
- Sogeti: A CapGemini consulting division that offers the Artificial Data Amplifier (ADA), an in-house solution that generates realistic data based on existing datasets.
- Statice: A platform that focuses on privacy preservation by generating fully synthetic twins. They also have an SDK for integration into existing pipelines. It includes support for the financial, healthcare and insurance industries.
- Synthesized: Synthesized is a development framework for creating synthetic data via API with a backend that supports integration with major relational databases, ETL and CLI tools, including some pre-generated public datasets.
- Syntho: Another SaaS that generates tabular data from an uploaded dataset that's able to handle time series data, geolocation data and sensitive characteristics.
- Tonic: A synthetic data platform that models existing datasets to produce a complete dataset or a subset of the data. Offers as a SaaS service and as a Docker container for on-prem deployment, which can cut privacy concerns when uploading data to a third party.

Unstructured data platforms are similar to game engines or film rendering engines, and, in fact, they enjoy a great deal of overlap with many of the technologies coming out of video games and film. They employ many of the same techniques as we see in those industries, like ray tracing, which can accurately recreate reflections of light by tracing light as it bounces around a scene. The reason unstructured data generation is further along is decidedly because photorealistic engines have been the goal of those two industries for decades, and they've largely achieved that goal.

- Anyverse: Creates synthetic data for self-driving cars, city planning and architectural planning and analysis.
- Bifrost: Creates perfectly realistic 3D worlds via API and auto generates the labels. Has the ability to generate rare events such as boats crashing into each other for predicting dangerous situations with computer vision.
- CVEDIA: Builds a platform that can simulate not just visual data but also sensor data, like LiDAR, radar, infrared, thermal and ultrasonic data.
- Cognata: Delivers digital twins or entirely synthetic worlds for AV testing and validation and can generate not just cities and roads but off-road data and agricultural plots for AV farm equipment, as well as realistic sensor models for uncommon sensors, such as radar, LiDAR and thermal cameras.
- Coohom Cloud: Offers a platform for generating synthetic indoor agents and robots.
- Deep Vision Data: Creates synthetic data for everything from manufacturing products to indoor and outdoor environments. One of the few companies that focuses on CAD systems and model defects or variations.
- Neurolabs: Focuses on synthetic data solutions for retail stores, such as generating realistic products and stocked shelves.
- Parallel domain: Platform for creating autonomous vehicle training, as well as sensor data for radar and LiDAR, with some large, high-tech customers.
- Pasteur iSi (Simulation Science): With state-of-art efficiency and fidelity, Pasteur builds in-silico playgrounds for human experts and AI agents to experiment with, simulating counterfactuals and understanding downstream effects before deployment.
- Rendered AI: Uses industry-standard models and techniques to create an unlimited variety of synthetic imagery for

AI training optimization.

- Scale: Scale AI, profiled in the labeling section of the report, also includes a beta tool called Scale Synthetic, for generating synthetic 2D and 3D data.
- Simerse: Focuses on manufacturing and construction data generation for inspectors of critical infrastructure and farm robots.
- Synthesis AI: Generates realistic humans for a variety of use cases like distracted driver monitoring or identity recognition. Users create JSON to outline desired data distributions and characteristics to submit jobs to their synthetic data cloud services job, while auto generating labels.

Current Trends and the Next Five Years

Synthetic data is poised to be a major accelerator of AI/ML in the coming decade, but it's currently in its infancy. Gathering data and labeling it are two of the biggest bottlenecks in creating effective models. Synthetic data has the potential to overcome both of them in one swoop.

That said, synthetic data is still a new entry into the AI/ML lifecycle, and it's still evolving. Most of the companies in the space started in the last two to four years. They are leveraging some of the most advanced research in ML to create artificial data. That's both a blessing and a curse because for artificial data generation to really advance, machine learning itself has to advance. Still, we've had enough evolution to create the possibility of artificial data and even at this early stage, it can make a difference in what organizations are trying to achieve now.

The most well-developed platforms in synthetic data come to us with a legacy from film, television and video games, which have enjoyed relentless advancement over the past thirty years to the point of ever-more photorealistic environments and people generation. Rendering engines and the tools to help developers design 2D and 3D worlds have reached an incredibly high level of photorealism, and it made sense to repurpose those platforms for synthetic data generation as a natural extension of their use cases. These tools enable skilled practitioners to address novel edge cases like creating images of manufacturing defects or demonstrating rare cases of boat collisions or rare driving conditions for autonomous vehicles.

Structured data solutions are still catching up to computer-vision use cases. The vast majority of the platforms surveyed offered robust abilities to create look-a-like datasets that protect privacy, which solves important challenges, like protecting customer health or financial data, but it doesn't yet solve some of the more pressing edge cases. Some of the techniques can create variance in structured data, but those variances won't showcase new or novel cases of fraud because the human creativity that goes into cheating the system is still beyond the capabilities of current AI techniques.

We advise anyone looking to use synthetic data to clearly understand what problem they are trying to solve and to research the abilities and limitations of synthetic data for their use case. For instance, if you are a financial institution looking to protect your customer data before training machine-learning models, you have some robust choices to pick from in the space. But if you're looking to invent novel fraud cases that have never existed, you will not find much to help you there as of yet.

To make a decision, outline your use cases clearly and then make a short list of teams that might meet those needs by looking carefully and critically at their offerings and the team that supports the platform.

The Next Five Years

We expect synthetic data to advance along with the general advancement of machine learning. With each new algorithm and technique to come out of the big AI research labs, we see synthetic data companies applying those advances in new and novel ways. In particular, structured data generation will continue to make strides because there are so many companies focused on it at the moment. We also expect to see an uptick of synthetic data generation to help address data quality issues by working hand in hand with supervision systems that can detect over- and underrepresentation and other more subtle data quality issues.

Structured data presents some interesting challenges in that it is harder to verify at the intuitive level the way you can verify visual imagery, so we expect the monitoring and QA of synthetic data to become an important part of structured data creation. We will start to see more automated ways of validating the generated data as part of the CI/CD pipeline, in the same way we see large suites of integrated and automated tests when we deploy code in DevOps pipelines.

While we've seen major advances in computer vision applications, we haven't seen as many advances in audio generation. It's probable that we'll see audio generators that are effective for synthetic data creation. We'll likely see different voices and languages automatically created for translations. We might also see the ability to add background noise or to automatically combine sounds together, which programmatically creates augmented datasets for audio.

That said, we don't necessarily need new use cases and capabilities for synthetic data to become a staple of the AI/ML workflow. We expect many organizations to adopt look-a-like datasets for privacy preservation over the next few years as regulators continue to create roadblocks to using personal information. This will be especially true in healthcare, drug discovery, finance and even in advertising. Creating a barrier between original datasets is likely to become standard practice in heavily regulated industries.

We expect to see a rise in both open-source offerings, which have lagged in this area of the ecosystem. Finally, we expect on-prem solutions to increase in the coming years. Since so much data is sensitive, uploading it to a third party presents challenges to teams looking to prevent privacy leaks, and on-premises solutions can fix that challenge. Lastly, on-premises solutions overcome the challenge of data gravity. Moving large datasets around via upload is time consuming and costly, and we expect to see more solutions that go right to where the data lives.

AI INFRASTRUCTURE ECOSYSTEM CONCLUSIONS AND WRAP UP

There's tremendous innovation happening in the AI/ML infrastructure space, but with so many platforms in the ecosystem, it's challenging for any one team to get their heads around all of it. This report was designed to bring clarity to this fascinating and rapidly developing space. Hopefully it gave you a solid foundation so you can now carefully examine the entire space to find the tools you need to grow your AI/ML teams and deliver more models into production faster.

As you saw, not all parts of the ecosystem are equally developed. Some parts are seeing rapid growth, like labeling accelerated by machine learning, while other parts are still developing and have more room to grow, like synthetic data. In some areas, like orchestration and pipelines, there is tremendous competition, while in other areas, like training, there are a few strong choices. Areas like AI supervision with monitoring, observability and explainability are seeing quick improvements and adoption.

The growth of any AI/ML team is a journey, and at each stage you need different tools. At any early stage, with only a few top-notch data scientists, your tooling needs are much simpler. But as your team grows, you need newer and better tools to deal with that growth. Traditional enterprise IT considerations, like role-based access control and security, suddenly become important, as does ongoing monitoring and maintenance. Larger teams may suddenly find themselves in need of a feature store as they put more and more models in production. Others discover they need data versioning and lineage. Some discover data versioning and lineage too late, after regulation or a public mistake highlights the need for it. In the early days, distributed training isn't much of a consideration, but as teams grow and compete for in-house resource scheduling across GPUs, it becomes essential. At each stage, new must-have tools rise to the surface rapidly.

As we've seen, the adoption of AI/ML is still in its early phase. Many big-tech companies built their own tools from scratch because there was nothing on the market to support their needs, but that approach is largely out of reach for other enterprises that don't have an army of developers. It's also unsustainable, as technical debt and maintenance of those tools quickly becomes a nightmare, even as commercial tools start to bypass internally built systems with their capabilities. We expect more and more tech companies to replace parts of their home-rolled stack with commercial or open-source alternatives in the next five years. We expect that most enterprises in the early majority stage will not craft their own tools and instead focus on writing smaller tools that close the gap between modular pieces of the stack.

In these early days, structured-data use cases are the most prevalent, and tools built on platforms like Spark are the most dominant. But we expect many of the most promising and bottom-line expanding use cases will come from unstructured data, deep learning, reinforcement learning and new techniques over the next decade. The next generation of tools that can harness the power of unstructured data and advanced use cases are still emerging, and there is no clear winner. Enterprises are just beginning to dip their toes into more advanced applications, but we expect that to expand quickly over the next five years as computer vision, NLP and other advanced applications become more well-known and accessible to teams that don't have dedicated researchers.

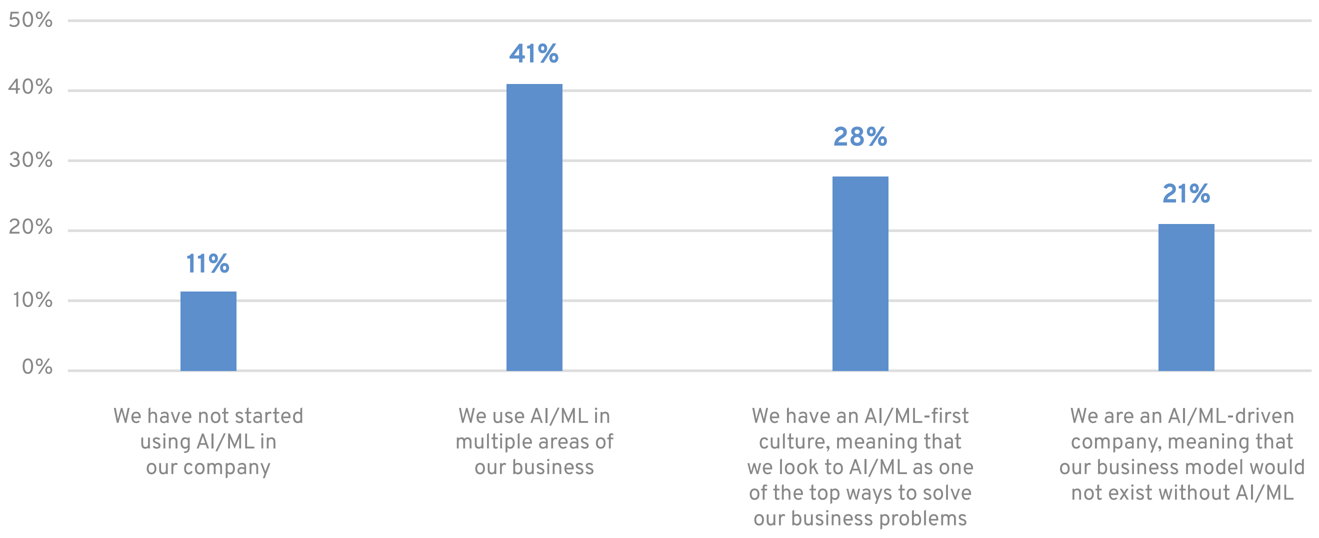
We also expect AI to get more generalized over the next five years, with single models capable of working with different modalities, such as images, video and text. Lastly, we expect to see models that move beyond the narrow and become capable of multiple tasks. We're already seeing the growth of multi-modal models like Google's LaMDA (Language Model for Dialogue Applications), along with OpenAI's DALLE 2 and CLIP.

To be very clear, we are not talking about AGI (artificial general intelligence). We are using the term generalized in a much more narrow sense, which we call GI or generalized intelligence. That is, AI that can move beyond a single, narrow task and do well across a single problem domain or a few domains. Research at multiple AI research labs and robotics institutes is testing multiple promising, practical approaches that can do multiple tasks like cleaning a house, opening and closing windows and folding clothes. We expect that research to accelerate and to ripple into pure software AI/ML applications that can do well across a broad domain.

We also expect that many smaller teams, with AI at the core of their business model, will continue to get funding. AI-driven businesses are businesses with AI at the very center of their business model, with capabilities that can only come from AI, such as photorealistic fashion model generators that provide virtual avatars to fashion houses to augment their online stores. That's just one example of thousands of AI-driven businesses that will emerge over the next five to ten years. These AI-driven businesses will grow and scale over the next decade to become some of the most important tech companies in the world as they outpace competitors who don't have AI expertise and can't compete without it. Other enterprises will adopt more and more advanced use cases to compete as these new tech titans emerge.

Those AI-driven businesses are already here. In our survey, 21% of companies count AI as central to their business model, and that number will only grow in the years to come.

How do you categorize your company?



AI will affect every single industry on earth, from telecommunications to finance to healthcare to defense to construction and manufacturing to agriculture to the automotive industry. There is not a single industry anywhere that won't benefit from more intelligence. But without the right infrastructure tools, organizations can't build those world changing applications. The AI/ML infrastructure stacks of today and tomorrow will form the foundation that allows the intelligence revolution to happen. Once those tools mature, the entire industry will move into the early majority stage, and we'll begin to see more and more applications that we simply can't live without and that we couldn't possibly create with traditional programming.

At the early majority stage, we expect to see consolidation among the players in the ecosystem. Some will fail. Some will merge. Some will get outflanked by competitors. But the companies that emerge from this early stage will form the canonical stack of AI/ML that enterprises everywhere rely on.

AI/ML infrastructure is the foundation on which to build the applications of tomorrow.

GLOSSARY

Terms in artificial intelligence are often used interchangeably, which creates tremendous confusion in the space. People use machine learning (ML) and artificial intelligence (AI) to mean the same thing, when in reality machine learning is a subset of artificial intelligence. Algorithms and models get swapped just as often, but algorithms create models, so they're not the same. We provide definitions for how we use these terms in this report below.

- **Artificial Intelligence =** Artificial Intelligence (AI) is a broad, overarching term for any software program that mimics the problem-solving and decision-making capabilities of the human mind.
- **Machine Learning =** Machine learning (ML) is a class of methods for automatically creating models from data. At its heart, ML is advanced pattern recognition.
- **Algorithm =** Algorithms are the engines of machine learning. They are procedures that run on data to create a machine learning model. It is the algorithm that learns from the data and outputs a model that can make inferences or predictions.
- **Model =** A model is a program that was trained to recognize patterns and make predictions.
- **Inference =** When a model makes predictions, we call those inferences.
- **Datum =** A single unit of a set of data, such as one cell in a set of rows and columns.
- **Supervised Learning =** Supervised learning uses labeled datasets to teach an algorithm what kind of features we want it to learn. We typically see supervised learning in the context of classification, when we want to map input to output labels, or regression, when we want to map input to a continuous output. Common algorithms in supervised learning include logistic regression, support vector machines, naive Bayes, neural networks, deep learning and random forests.
- **Semi-Supervised Learning =** Semi-supervised learning looks to use a small set of labeled training data together

with a larger amount of unlabeled training data. The need for semi-supervised approaches comes from real-world situations in which labeling data is incredibly time consuming or impossible and/or very expensive and/or involves a constant stream of data. If we were trying to detect hate speech on a social media network, there's simply no way to hand annotate each message that comes in because there are too many messages to go through individually. Instead, we can use semi-supervised approaches where we hand label a subset of messages and leverage those to help us understand the rest of the messages as they come in.

- **Unsupervised Learning =** Unsupervised learning is a type of algorithm that learns patterns from unlabeled data. Through mimicry, the software builds a compact internal representation of its world and then generates predictions from it. Some of the most common algorithms used in unsupervised learning include clustering, anomaly detection, and various approaches for learning latent variable models.

- **Reinforcement Learning =** Reinforcement learning (RL) trains a software agent, through game-like environments, to achieve a goal in an uncertain and complex situation. The agent goes through a series of trial and answer solutions, and the system rewards or punishes the agent's approach to achieving its goal. The agent looks to maximize its total reward.

- **Structured Data =** Structured data is data that fits into a column and row structure, such as a relational database. It adheres to a pre-build schema. Examples include names, dates, credit card numbers and time series data.

- **Unstructured Data =** Unstructured data has no fixed data schema. Examples of unstructured data are videos, audio files, images and unstructured text like tweets, legal documents, novels and reports.

- **Semi-Structured Data =** Semi-structured data is a kind of data that has some regularity to it but can often include unstructured text elements, such as JSON or YAML.

Website

ai-infrastructure.org

LinkedIn

linkedIn.com/company/ai-infrastructure-alliance

Twitter

twitter.com/AIInfra

AI Infrastructure Alliance