Scaling Machine Learning at Uber with Michelangelo | Uber Engineering Blog

Scaling Machine Learning at Uber with Michelangelo

In September 2017, we published an article introducing Michelangelo, Uber’ s Machine Learning Platform, to the broader technical community. At that point, we had over a year of production experience under our belts with the first version of the platform, and were working with a number of our teams to build, deploy, and operate their machine learning (ML) systems.

As our platform matures and Uber’ s services grow, we’ ve seen an explosion of ML deployments across the company. At any given time, hundreds of use cases representing thousands of models are deployed in production on the platform. Millions of predictions are made every second, and hundreds of data scientists, engineers, product managers, and researchers work on ML solutions across the company.

In this article, we reflect on the evolution of ML at Uber from the platform perspective over the last three years. We review this journey by looking at the path taken to develop Michelangelo and scale ML at Uber, offer an indepth look at Uber’ s current approach and future goals towards developing ML platforms, and provide some lessons learned along the way. In addition to the technical aspects of the platform, we also look at the important organizational and process design considerations that have been critical to our success with ML at Uber.

Zero to 100 in three years

In 2015, ML was not widely used at Uber, but as our company scaled and services became more complex, it was obvious that there was opportunity for ML to have a transformational impact, and the idea of pervasive deployment of ML throughout the company quickly became a strategic focus.

While the goal of Michelangelo from the outset was to democratize ML across Uber, we started small and then incrementally built the system. Michelangelo’ s initial focus was to enable large-scale batch training and productionizing batch prediction jobs. Over time, we added a centralized feature store, model performance reports, a low-latency real-time prediction service, deep learning workflows, notebook integrations, partitioned models, and many other components and integrations.

In three short years, Uber went from having no centralized ML efforts and a few bespoke ML systems to having advanced ML tools and infrastructure, and hundreds of production ML use-cases.

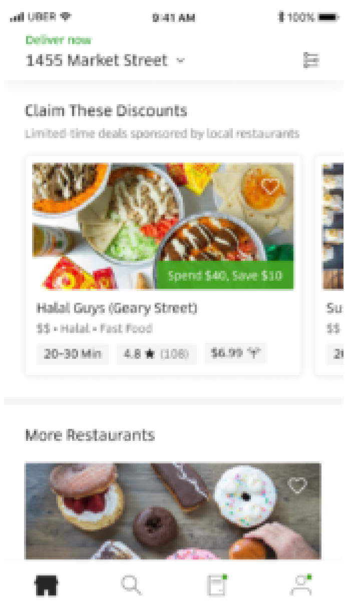
ML use cases at Uber

Uber uses ML for a very diverse set of applications. Rather than applying ML to a few key areas (such as ad optimization or content relevance), Uber has a much more even spread of ML solutions. In this section, we discuss a select few Michelangelo use cases that came up over the last three years, highlighting the diversity and impact of ML at Uber:

Uber Eats

Uber Eats uses a number of machine learning models built on Michelangelo to make hundreds of predictions that optimize the eater experience each time the app is opened.

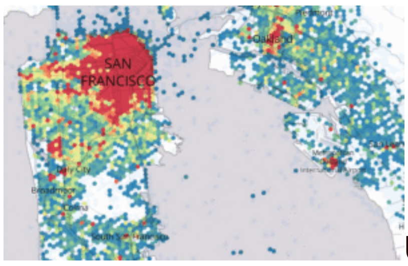
ML-powered ranking models suggest restaurants and menu items based on both historical data and information from the user’ s current session in



the app (e.g. their search query).

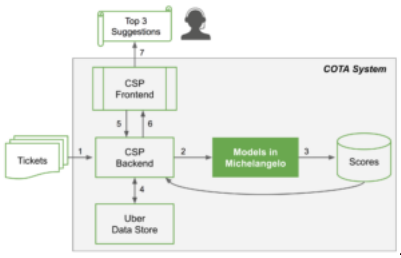
Using Michelangelo, Uber Eats also estimates meal arrival times based on predicted ETAs, historical data, and various real-time signals for the meal and restaurant.

Marketplace Forecasting



Uber’ s Marketplace team leverages a variety of spatiotemporal forecasting models that are able to predict where rider demand and driver-partner availability will be at various places and times in the future. Based on forecasted imbalances between supply and demand, Uber systems can encourage driver-partners ahead of time to go where there will be the greatest opportunity for rides.

Customer Support

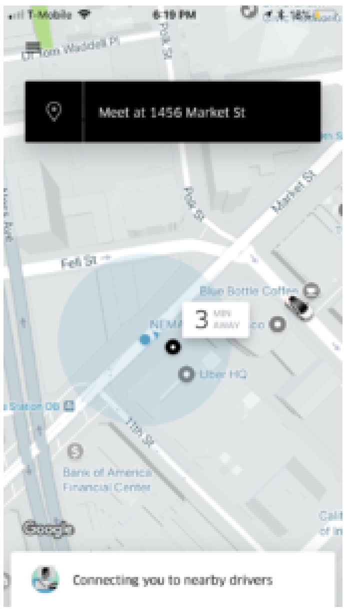


Around 15 million trips happen on Uber every day. People frequently leave wallets or phones in the car or have other problems that lead to thousands of support tickets each day through our help system. These tickets are routed to customer service representatives. Machine learning models built in Michelangelo are heavily used to automate or speed-up large parts of the process of responding to and resolving these issues. The first version of these models, based on boosted trees, sped up ticket handling time by 10 percent with similar or better customer satisfaction. The second version, based on a deep learning model, drove an additional 6 percent speedup.

Ride Check

Since the very first Uber ride in 2010, GPS data has been used to put every trip on the map so we know where and when you’ re riding and who’ s behind the wheel. But we can do more: by harnessing the power of GPS and other sensors in the driver’ s smartphone, our technology can detect possible crashes. This technology can also flag trip irregularities beyond crashes that might, in some rare cases, indicate an increased safety risk. For example, if there is a long, unexpected stop during a trip, both the rider and the driver will receive a notification through our Ride Check feature that offers assistance in the event of a crash.

Estimated Times of Arrival (ETAs)



One of the most important and visible metrics for the company are ETAs for rider pickups. Accurate ETAs are critical to a positive user experience, and these metrics are fed into myriad other internal systems to help determine pricing and routing. However, ETAs are notoriously difficult to get right.

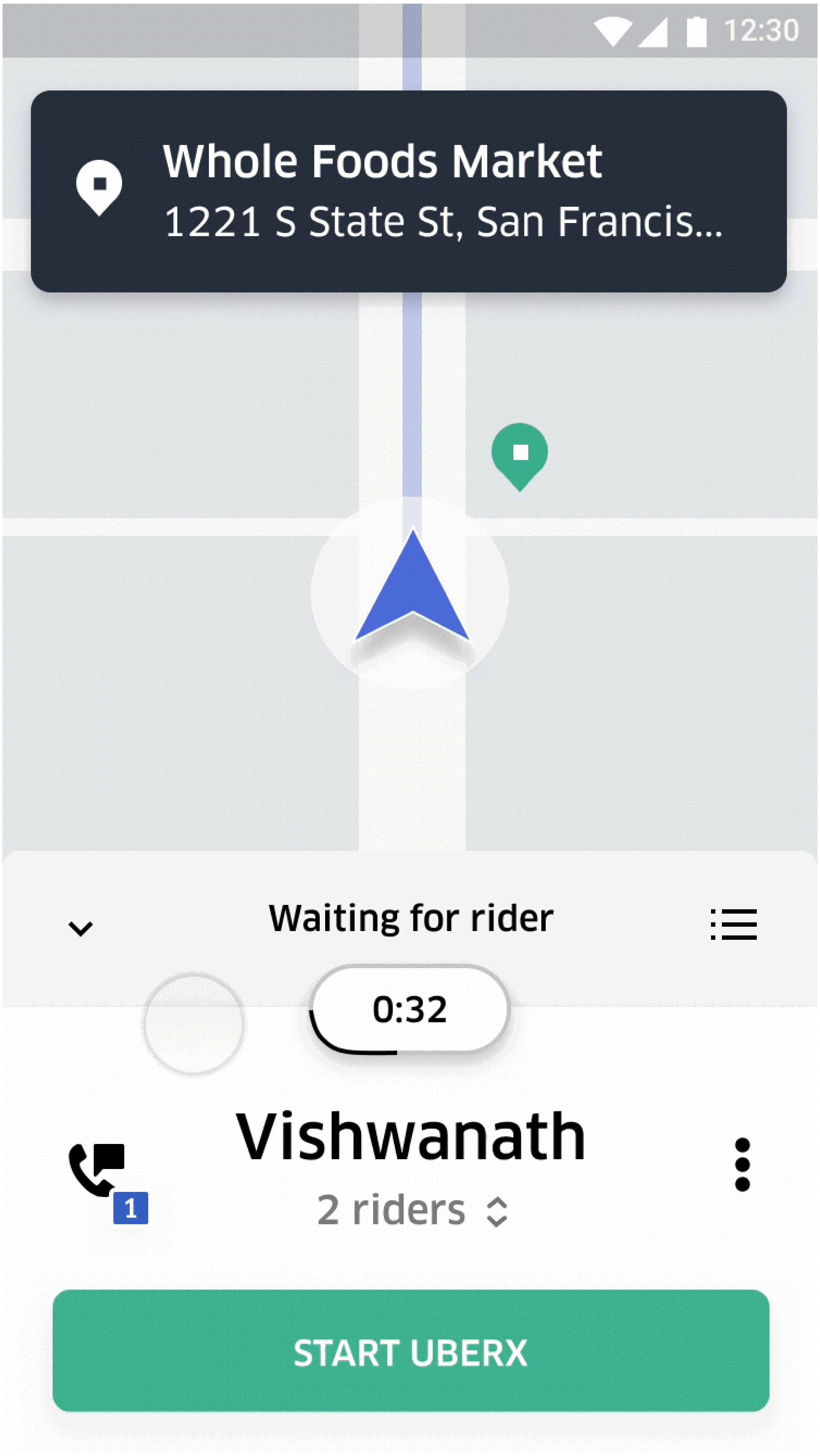
Uber’s Map Services team developed a sophisticated segment-by-segment routing system that is used to calculate base ETA values. These base ETAs have consistent patterns of errors. The Map Services team discovered that they could use a machine learning model to predict these errors and then use the predicted error to make a correction. As this model was rolled out city-by-city (and then globally for the last couple of years), we have seen a dramatic increase in the accuracy of the ETAs, in some cases reducing average ETA error by more than 50 percent.

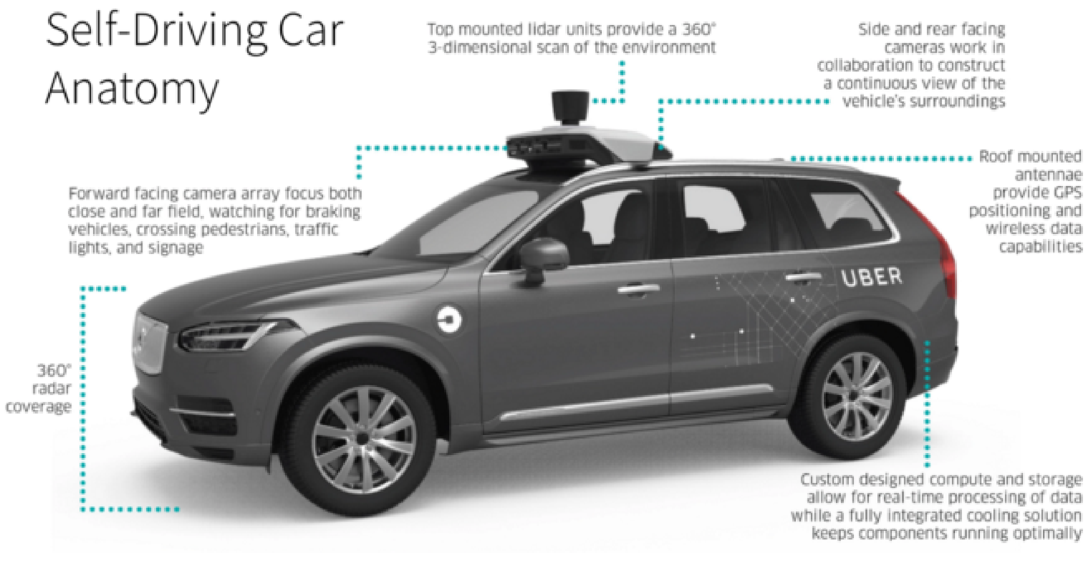
One-Click Chat

The one click chat feature streamlines communication between riders and driver-partners by using natural language processing (NLP) models that predict and display the most likely replies to in-app chat messages. Letting driver-partners respond to rider chat messages with a single button press reduces distraction.

Self-Driving Cars

Uber’s self-driving car systems use deep learning models for a variety of functions, including object detection and motion planning. The modelers use Michelangelo’s Horovod for efficient distributed training of large models across a large number of GPU machines.


12:30
Whole Foods Market
1221 S State St, San Francis...
Waiting for rider
0:32
Vishwanath
2 riders
1
START UBERX


Self-Driving Car Anatomy
Top mounted lidar units provide a 360° 3-dimensional scan of the environment
Side and rear facing cameras work in collaboration to construct a continuous view of the vehicle's surroundings
Roof mounted antennae provide GPS positioning and wireless data capabilities
Forward facing camera array focus both close and far field, watching for braking vehicles, crossing pedestrians, traffic lights, and signage
360° radar coverage
UBER
Custom designed compute and storage allow for real-time processing of data while a fully integrated cooling solution keeps components running optimally

How we scaled ML at Uber

As a platform team, our mission is to unlock the value of ML and accelerate its adoption in all corners of the company. We do this by democratizing the tools and support our technical teams need, namely, optimizing for developer velocity, end-to-end ownership, software engineering rigor, and system flexibility.

For data scientists, our tooling simplifies the production and operations side of building and deploying ML systems, enabling them to own their work end-to-end. For engineers, Uber’ s ML tooling simplifies the data science (feature engineering, modeling, evaluation, etc.) behind these systems, making it easy for them to train sufficiently high-quality models without needing a data scientist. Finally, for highly experienced engineering teams building specialized ML systems, we offer Michelangelo’ s ML infrastructure components for customizable configurations and workflows.

Successfully scaling ML at a company like Uber requires getting much more than just the technology right—there are important considerations for organization and process design as well. In this section, we look at critical success factors across three pillars: organization, process, as well as technology.

Figure 1: The core strategy pillars of the Michelangelo Machine Learning Platform.

Organization

Widely varying requirements for ML problems and limited expert resources make organizational design particularly important—and challenging—for machine learning. While some ML projects at Uber are owned by teams with multiple ML engineers and data scientists, others are owned by teams with little to no technical expertise. Similarly, some problems can be solved by novices with commonly available out-of-the-box algorithms, while other problems require expert investigation with advanced techniques (and often don’t have known solutions).

Getting the right people working on the right problems has been critical to building high quality solutions and deploying them consistently and successfully in production. The challenge is in allocating scarce expert resources and amplifying their impact across many different ML problems. For example, if a new project requires computer vision know-how, what organizational structure will allow Uber to effectively allocate expert resources in a way that is aligned with company priorities?

After several iterations, Uber currently operates with the following main roles and responsibilities:

Figure 2: Organizational interactions of different teams in Uber’s ML ecosystem.

Let’s take a look at some of the key teams and how they work together to design, build, and deploy new ML systems in production.

Product teams

We found that it works best if the product engineering teams own the models they build and deploy in production. For example, our Map Services team owns the models that predict Uber’s ETAs. Product teams are typically staffed with the full set of skills they need to build and deploy models using Uber’s ML platforms. When they need additional expertise, they get assistance from the research and/or specialist teams.

Product organizations sometimes also have special teams who help address any gaps between what the platform provides and what specific product engineering teams need. These teams adapt the centralized platform tools for their use case and fill in feature gaps with tailored tools and workflows. For instance, many teams in Uber’s Marketplace organization have similar workflows around training, evaluating, and deploying models per city and product. A Marketplace team creates specialized tools that sit on top of Michelangelo, making it easier to manage these Marketplace ML projects.

Specialist teams

When product engineering teams encounter ML problems that stretch their abilities or resources, they can turn to an internal team of specialists for help. Uber’ s specialists have deep expertise across different domains—like NLP, computer vision, recommender systems, forecasting—and partner with product engineering teams to build tailored solutions. For instance, our COTA project is an effort that pairs a specialist team with a product team to create massive impact for our business and customers.

Typically, these projects last a few weeks to many quarters. As a project is de-risked and moves closer to launching in production, product teams often add relevant full-time experts to fill the expertise gap, ensure they’ re able to maintain the system on their own, and free up specialist resources.

Research teams

Specialists and product engineering teams often engage with Uber’ s AI research group, AI Labs, to collaborate on problems and help guide the direction for future research. Research teams typically do not own production code, but they frequently work closely with different teams on applied problems. When relevant new techniques and tools are developed by researchers, the platform engineering team integrates them into company-wide platforms, allowing new techniques to be easily leveraged across the company.

ML Platform teams

The Michelangelo Platform team builds and operates a general purpose ML workflow and toolset that is used directly by the product engineering teams to build, deploy, and operate machine learning solutions.

As our systems become more sophisticated and the problems we solve more complex, demand grows for additional flexibility, extensibility, and domain-specific ML development experiences. We’ re spinning up a number of other, more domain-specific platforms to address specialized use cases that are not as well served by Michelangelo workflow tools. These new platform teams reuse a lot of the existing Michelangelo platform and deliver specialized ML development workflows to product teams. For instance, there are NLP and computer vision-specific platforms being built that contain special visualization tools, pre-trained models, metadata tracking, and other components that don’t fit well in a general-purpose platform.

Process

As Uber’s ML operations mature, a number of processes have proven useful to the productivity and effectiveness of our teams. Sharing ML best practices (e.g., data organization methods, experimentation, and deployment management) and instituting more structured processes (e.g., launch reviews) are valuable ways to guide teams and avoid repeating others’ mistakes. Internally focused community building efforts and transparent planning processes engage and align ML teams under common goals.

Launching models

Designing reliable processes to avoid common development pitfalls and verify intended model behavior are critical to safely scaling ML in an organization. ML systems are particularly vulnerable to unintended behaviors, tricky edge cases, and complicated legal/ethical/privacy problems. In practice, however, risk profiles differ significantly across use cases and require tailored approval and launch processes. For example, launching an automated update to an ETA prediction model that uses anonymized data requires less privacy scrutiny than launching a new pricing model.

For these reasons, product organizations (e.g., the Uber Eats or Marketplace teams) own the launch processes around their ML models. These teams adapt processes to their product area from a centralized launch playbook that walks through general product, privacy, legal, and ethical topics around experimenting with and launching ML models. The product teams themselves best understand the product implications of different model behavior and are best suited to consult with relevant experts to evaluate and eliminate risks.

Coordinated planning across ML teams

When requirements outpace the roadmaps of the platform teams, product engineering teams can feel the desire to branch off and build their own systems tailored to their needs. Care needs to be taken to ensure teams are empowered to solve their own problems but also that the company is making good engineering tradeoffs to avoid fragmentation and technical debt. At Uber, we put together an internal group of senior leaders that oversees the evolution of ML tooling across the company to ensure that we’ re making smart trade-offs and are maintaining long-term architecture alignment. This has been invaluable in resolving these tricky and sometimes sensitive situations.

Community

Scaling high-quality ML across the company requires a connected and collaborative organization.

To build an internal community, we host an annual internal ML conference called UberML. We recently hosted around 500 employees and more than 50 groups presenting talks or posters on their work. Events like this enable practitioners to swap ideas, celebrate achievements, and make important connections for future collaborations. Teams at Uber also organize community building events including ML reading groups, talk series, and regular brown bag lunches for Uber’ s ML-enthusiasts to learn about some of our internal ML projects from the individuals that build them.

Our focus on community extends beyond our own walls. Our team also engages heavily with the external ML community through conferences, publishing papers, contributing to open source projects, and collaborating on ML projects and research with other companies and academia. Over the years, this community has grown into a global effort to share best practices, collaborate on cutting-edge projects, and generally improve the state of the field.

Education

It’ s important for ML teams to always be learning. They need to stay on top of developments in ML theory, track and learn from internal ML projects, and master the usage of our ML tools. Proper channels to efficiently share information and educate on ML-related topics are critical.

Uber ML education starts during an employee’ s first week, during which we host special sessions for ML and Michelangelo boot camps for all technical hires. When major new functionality is released in Michelangelo, we host special training sessions with the employees that frequently use them. Documentation of key tools and user workflows has also helped encourage knowledge sharing and scaled adoption of our platform tools.

Office hours are also held by different ML-focused groups in the company to offer support when questions arise. It also helps that the individuals who work on ML projects at Uber tend to be naturally inquisitive and hungry learners. Many of the community-led initiatives mentioned above are great ways for team members to keep up with internal and external developments.

Technology

There are myriad details to get right on the technical side of any ML system. At Uber, we’ ve found the following high-level areas to be particularly important:

- - **End-to-end workflow**: ML is more than just training models; you need support for the whole ML workflow: manage data, train models, evaluate models, deploy models and make predictions, and monitor predictions.
  - **ML as software engineering**: We have found it valuable to draw analogies between ML development and software development, and then apply patterns from software development tools and methodologies back to our approach to ML.
  - **Model developer velocity: Machine learning model development is a very iterative process—innovation and high-quality models come from lots and lots of experiments. Because of this, model developer velocity is critically important.**
  - **Modularity and tiered architecture**: Providing end-to-end workflows is important for handling the most common ML use cases, but to address the less common and more specialized cases, it’ s important to have primitive components that can be assembled in targeted ways.

End-to-end workflow

Early on, we recognized that successful ML at a large company like Uber requires much more than just training good models—you need robust, scalable support for the entire workflow. We found that the same workflow applies across a wide array of scenarios, including traditional ML and deep learning; supervised, unsupervised, and semi-supervised learning; online learning; batch, online, and mobile deployments; and time-series forecasting. It’ s not critical that one tool provides everything (though this is how we did it) but it is important to have an integrated set of tools that can tackle all steps of the workflow.

Manage data

This is typically the most complex part of the ML process and covers data access, feature discovery, selection, and transformations that happen during model training and the productionization of pipelines for those features when the model is deployed. At Uber, we built a feature store which allows teams to share high-quality features and easily manage the offline and online pipelines for those features as models are trained and then deployed, ensuring consistency between online and offline versions.

Train models

In Michelangelo, users can train models from our web UI or from Python using our Data Science Workbench (DSW). In DSW, we support large-scale distributed training of deep learning models on GPU clusters, tree and linear models on CPU clusters, and lower scale training of a large variety of models using the myriad available Python toolkits. In addition to training simple models, users can compose more complex transformation pipelines, ensembles, and stacked models. Michelangelo also offers scalable grid and random hyperparameter search, as well as more efficient Bayesian black-box hyperparameter search.

Manage and evaluate models

Finding the right combination of data, algorithm, and hyperparameters is an experimental and iterative process. Moving through this process quickly and efficiently requires automation of all the experiments and the results. It also benefits from good visualization tools for understanding each individual model’ s performance as well as being able to compare many models with each other to see the patterns of configuration and feature data that improve the model performance. Models managed in Michelangelo are rigorously managed, version controlled, fully reproducible, and have rich visualizations for model accuracy and explainability.

Figure 3: Michelangelo’ s model comparison page showing a comparison of two models’ behavior across different segments and features.

Deploy models and make predictions

Once an effective model is trained, it’ s important for the model developer to be able to deploy the model into a staging or production environment. In Michelangelo, users can deploy models via our web UI for convenience or through our API for integration with external automation tools. At deploy time, the model and related resources are packed up and then pushed out to an offline job for scheduled batch predictions or to online containers for real-time request-response predictions via Thrift. For both online and offline models, the system automatically sets up the pipelines for data from the feature store.

Monitor data and predictions

Models are trained and initially evaluated against historical data. This means that users can know that a model would have worked well in the past. But once you deploy the model and use it to make predictions on new data, it’ s often hard to ensure that it’ s still working correctly. Models can degrade over time because the world is always changing. Moreover, there can be breakages or bugs in a production model’ s data sources or data pipelines. In both cases, monitoring of (and alerting on) predictions made by models in production is critical. We have two approaches to monitoring models in production. The most accurate approach is to log predictions made in production and then join these to the outcomes as they are collected by our data pipelines; by comparing predictions against actuals, we can compute precise accuracy metrics. In cases where the outcomes are not easily collected or where we cannot easily join the predictions to outcomes, a second option is to monitor the distributions of the features and predictions and compare them over time. This is a less precise approach, but can still often detect problematic shifts in features and corresponding predictions.

ML as software engineering

An important principle of the Michelangelo team’ s approach is to think of *machine learning as software engineering***.** Developing and running ML in production should be as iterative, rigorous, tested, and methodological as software engineering. We have found it very valuable to draw analogies between ML and software development, and to apply insights from corresponding and mature software development tools and methodologies back to ML.

For instance, once we recognized that a model is like a compiled software library, it becomes clear that we want to keep track of the model’ s training configuration in a rigorous, version controlled system in the same way that you version control the library’ s source code. It has been important to keep track of the assets and configuration that were used to create the model so that it can be reproduced (and/or improved) later. In the case of transfer learning in deep learning models, we track the entire lineage so that every model can be retrained, if needed. Without good controls and tools for this, we have seen cases in which models are built and deployed but are impossible to reproduce because the data and/or training configuration has been lost.

In addition, to make sure software works correctly, it is important to run comprehensive tests before the software is deployed; in the same way, we always evaluate models against holdout sets before deploying. Similarly, it is important to have good monitoring of software systems to make sure they work correctly in production; the same applies to machine learning where you want to monitor the models in production as they may behave differently than they did in offline evaluation.

Model developer velocity

Building impactful ML systems is a science and requires many iterations to get right. Iteration speed affects both how ML scales out across the organization and how productive a team can be on any given problem. A high priority for the Michelangelo team is enabling data science teams to go faster. The faster we go, the more experiments we can run, the more hypotheses we can test, the better results we can get.

The diagram below shows how we think about the standard ML development process and the different feedback loops within them. We are constantly thinking about this process and tightening these loops so it’s easier and faster to do iterative and agile data science.

Figure 4: The workflow of a machine learning project. Defining a problem, prototyping a solution, productionizing the solution and measuring the impact of the solution is the core workflow. The loops throughout the workflow represent the many iterations of feedback gathering needed to perfect the solution and complete the project.

Michelangelo’s “zero-to-one speed” or “time-to-value speed” is critical for how ML spreads across Uber. For new use cases, we focus on lowering the barrier to entry by fine-tuning the getting started workflow for people of different abilities and having a streamlined flow to get a basic model up and running with good defaults.

For existing projects, we look at iteration speed**,** which gates how fast data scientists can iterate and get feedback on their new models or features either in an offline test or from an online experiment.

A few principles have proven very useful in enabling teams to develop quickly:

1. Solve the data problem so data scientists don’ t have to.
   - Dealing with data access, integration, feature management, and pipelines can often waste a huge amount of a data scientist’ s time. Michelangelo’ s feature store and feature pipelines are critical to solving a lot of data scientist headaches.

2. Automate or provide powerful tools to speed up common flows.
3. Make the deployment process fast and magical.
   - Michelangelo hides the details of deploying and monitoring models and data pipelines in production behind a single click in the UI.

4. Let the user use the tools they love with minimal cruft— “Go to the customer” .
   - Michelangelo allows interactive development in Python, notebooks, CLIs, and includes UIs for managing production systems and records.

5. Enable collaboration and reuse.
   - Again, Michelangelo’ s feature store is critical to enabling teams to reuse important predictive features already identified and built by other teams.

6. Guide the user through a structured workflow.

Going to the customer: Notebooks and Python

When Michelangelo started, the most urgent and highest impact use cases were some very high scale problems, which led us to build around Apache Spark (for large-scale data processing and model training) and Java (for low latency, high throughput online serving). This structure worked well for production training and deployment of many models but left a lot to be desired in terms of overhead, flexibility, and ease of use, especially during early prototyping and experimentation.

To provide greater flexibility, ease of use and iteration speed, we are moving the main model building workflows to Uber’ s DSW. DSW provides flexible and easy access to Uber’ s data infrastructure and compute resources in a natural notebook interface. Its integration with our cloud and on-prem GPU clusters allows for fast prototyping of Michelangelo-ready ML models in a notebook environment and easy saving of those models in Michelangelo for deployment and scaled serving. We’ re transitioning to using DSW as the primary model exploration and prototyping interface for Michelangelo.

To support the same scalable modeling in a notebook environment that we have always provided via our UI, we have released (internally for now, but we hope to open source shortly) a set of libraries that extend Spark to provide a set of custom *Estimator*, *Transformer*, and *Pipeline* components that expose interfaces for batch, streaming, and request/response-based scoring (the latter is not available in the standard version of Spark). These components can be assembled using PySpark and then uploaded to Michelangelo for deployment and serving using our pure-Java serving system. This brings together much of the ease of use of Python with the scale of Spark and Java.

Plain Python modeling is advantageous for simplicity and access to a richer ecosystem of ML and data toolkits. To address this, we recently expanded Michelangelo to serve any kind of Python model from any source for more flexible modeling support. Users build their models in DSW notebooks (or other preferred Python environment) and then use the Michelangelo PyML SDK to package and upload the model and dependencies to Michelangelo for storage, deployment, and serving (both batch and online).

Speed with deep learning

The development workflow for deep learning models often has different requirements than other ML development workflows. Developers typically write a lot more detailed training code and require specialized compute resources (GPUs). We’ ve focused a lot on making this process smooth and fast over the past year.

Michelangelo now has great tools to provision and run training jobs on different GPU machines both in Uber’ s own data centers and various public clouds. Production TensorFlow models are served out of our existing high-scale Michelangelo model serving infrastructure (which is now integrated with TensorFlow Serving) or our PyML system. We have specialized tools to help modelers track their experiments and development, but once a model is saved in Michelangelo, it’ s treated just like any other model in the system.

Speeding up model development with AutoTune

AutoTune is a new general purpose optimization-as-a-service tool at Uber. It has been integrated into Michelangelo to allow modelers to easily use state-of-the-art black box Bayesian optimization algorithms to more efficiently search for an optimal set of hyperparameters. It serves as a new recommended alternative to the less sophisticated search algorithms that we’ ve been offering in Michelangelo so far. This means more accurate models in the same amount of training time or less training time to get to a high-quality model.

Modularity and tiered offerings

One of the tensions we found while developing Michelangelo was between providing end-to-end support for the most common ML workflows while also providing the flexibility to support the less common ones.

Originally, our platform and infrastructure components were combined into a single system. As our systems became more sophisticated and the problems we were solving became more varied and complex, demand grew for additional flexibility, extensibility, and domain-specific development experiences beyond what a monolithic platform could offer.

We were able to address some of these issues through the bridge teams, as described above. But some teams wanted to mix and match parts of Michelangelo with their own components into new workflows. Other teams needed specialized development tools for their use cases but it didn’ t make sense to build those tools from scratch. We made some major changes to the Michelangelo architecture to leverage our existing systems as much as possible while evolving with growing requirements as ML matured across the company:

We also found that for some problem domains, specialized development experiences are useful. This can be as simple as prebuilt workflows for applying and evaluating forecasting models or it can be something more customized, like an interactive learning and labeling tool built for a particular computer vision application. We want to support all of these use cases by allowing platform developers to leverage Michelangelo’ s underlying infrastructure.

To address these issues, we are in the process of factoring out Michelangelo’ s infrastructure into an explicit infrastructure layer and are making that infrastructure available for teams to leverage to build more specialized platforms, for example, for NLP or Vision. Once this is done, we will have two customer groups: the model builders who use the Michelangelo platform to build and deploy models, and ML systems builders who use the Michelangelo infrastructure components to build bespoke solutions or more specialized platforms.

Key lessons learned

Building Michelangelo and helping to scale machine learning across Uber over the last three years, we have learned a lot from our successes and failures. In some cases, we got things right the first time, but more frequently, it took a few iterations to discover what works best for us.

- **Let developers use the tools that they want.** This is an area where it took us several iterations over as many years to figure out the right approach. When we started Michelangelo, we focused first on the highest scale use cases because this was where we could have the most impact. While the early focus on Scala, config, and UI-based workflows allowed us to support the high scale use cases, it led us away from the mature, well-documented programmatic interfaces that model developers are already proficient with. Since modeling work is very iterative, it ended up being important to focus on developer velocity (and therefore fast iteration cycles) along with high scalability. We have landed on a hybrid solution where we offer a high scale option that is a bit harder to use and a lower scale system that is very easy to use.
- **Data is the hardest part of ML and the most important piece to get right.** Modelers spend most of their time selecting and transforming features at training time and then building the pipelines to deliver those features to production models. Broken data is the most common cause of problems in production ML systems. At Uber, our feature store addresses many of these issues by allowing modelers to easily share high-quality features, automatically deploy the production pipelines for those features, and monitor them over time. Once features are defined, we can also leverage Uber’ s data quality monitoring tools to ensure that the feature data is correct over time.
- **It can take significant effort to make open source and commercial components work well at scale.** Apache Spark and Cassandra are both popular and mature open source projects; however, it took more than a year of sustained effort in each case to make them work reliably at Uber’ s scale. We had similar challenges with commercial toolkits that we tried early on and ended up abandoning.
- **Develop iteratively based on user feedback, with the long-term vision in mind.** As we built Michelangelo, we almost always worked closely with customer teams on new capabilities. We would solve the problem well for one team first, and once it was successfully running in production, we’ d generalize for the rest of the company. This process

ensured that the solutions we built were actually used. It also helped keep the team engaged and leadership supportive because they saw steady impact. At the same time, it’ s important to reserve some time for long-term bets that users may not see on their horizon. For instance, we started working on deep learning tooling well before there was real demand; if we had waited, we would have been too late.

- **Real-time ML is challenging to get right.** Most existing data tools are built for offline ETL or online streaming. There are no great tools–yet!–that address the hybrid online/offline capabilities (batch, streaming, and RPC) required by realtime ML systems. This continues to be a big area of focus for us at Uber as part of our feature store.

The journey is just beginning and there’ s still much to do. This is an evolving space and there will be many more lessons to learn. If you are interested in tackling machine learning challenges that push the limits of scale, consider applying for a role on our team!

Jeremy Hermann and Mike Del Balso are the Engineering and Product Managers on Uber’ s Machine Learning Platform team.

Comments