tecton

How to Make the Jump from Batch to Real-Time Machine Learning

Live Q&A

Q. Won't loan transactions take time and be processed later instead of real-time?

Not necessarily. In Tide's case, for example, they do the majority of loan requests in near real-time (<1 second).

Q. What is the approximate percentage of use cases today that are real-time vs. batch?

It's difficult to say. Based on our customer interactions, <20% of use cases are real-time.

Q. Has real-time ML known much success in fighting bot-driven fraud, for example, sites using bots to drive marketing referral payments?

Yes. In fact, there's a fascinating talk by LinkedIn in which they show how changing feature freshness from hourly to minutes helped them catch ~30% more bots.

Q. Do you have a mental framework or any use case you can share for healthcare where camera equipment or different demographics could be a potential constraint for data drift?

Healthcare is pretty tricky because of sensitive data and data mobility (for example, you don't want to send the data from one hospital to another) because of the number of different devices involved, and because most decisions need to be made quickly.

So not only do systems need to process information and make decisions quickly, but in most cases, a device must process data locally. One of Tecton's customers, Vital, is a startup in the healthcare space

that developed an application that predicts wait times for a bed or triage for patients coming into the emergency room. It's a simple real-time machine learning application that yields real impact for end-users. The application is not using camera equipment, but it is a real-time predictive application in the healthcare space. You can learn more about Vital's customer-facing emergency department (ED) predictive product here.

Q. Real-time machine learning or real-time streaming ML systems can be challenging to interpret and explain. This can be problematic in applications where explainability is important like healthcare and finance. How does one tackle this problem?

This problem exists across machine learning and depends on the model. For example, decision trees are significantly more straightforward to interpret than deep learning models with millions of layers. Therefore, it really comes down to a case-by-case decision that depends on regulatory or even ethical constraints. If full interpretability is a requirement, some models may just be out of the question, and falling back on a decision tree may be the best way forward.

Q. There may be errors or noise in real-time data. How can I solve this when making an online prediction with real-time ML?

You need to do more than just monitor your models at prediction time. You also need to monitor your features to prevent a "garbage in, garbage out" situation. However, it's tough to detect problems with the data being served to your models. This is especially true for real-time production ML applications like recommender or fraud detection systems. You can read more about feature monitoring for real-time machine learning here.

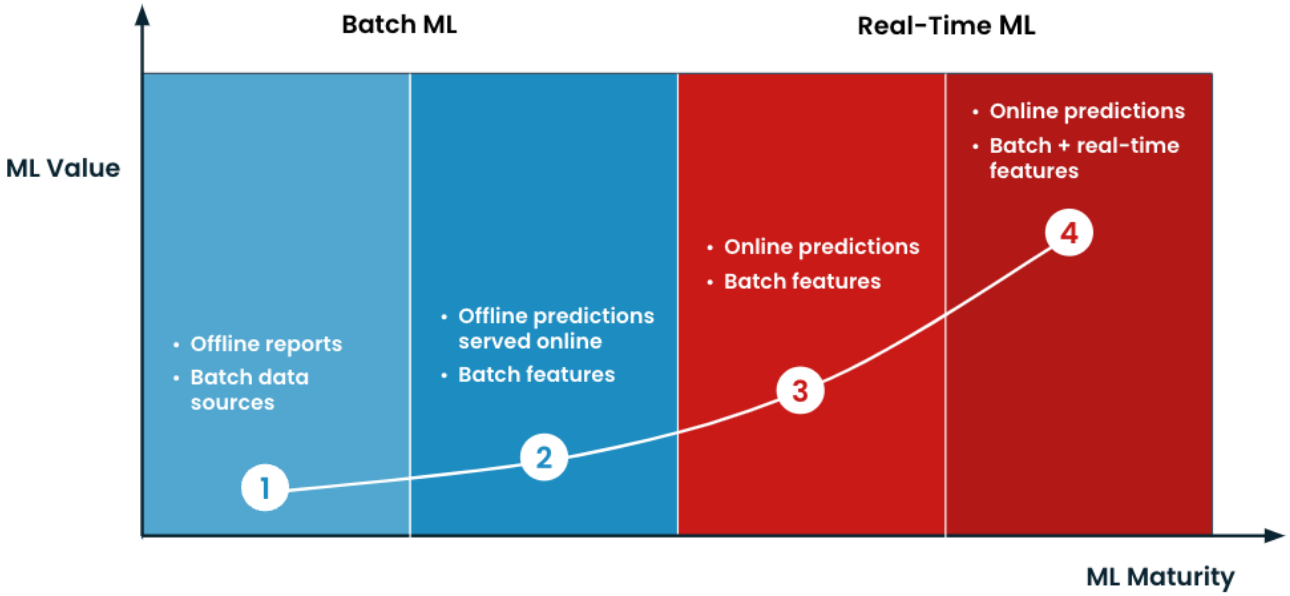
Q. Is it necessary to use streaming (data) systems to use real-time features? Is this the only way to get them?

The short answer is no. You can get real-time features by processing data on demand coming directly from an application via API. We've designed Tecton to support this setup as well.

Q. I find the difficult part is getting some of the batch features into the edge device (smart meters, for example) to enable stage 4 (real-time ML with online predictions and batch + real-time features).

This maturity graph doesn't differentiate between edge machine learning and server machine learning, where a client still talks to the server. For simplicity's sake, we've put them all into one box. In an ideal world, with edge machine learning applications, systems have the luxury of talking to a server that can

execute heavier processing and typically has access to more signals. But in some cases, depending on the use case at hand, systems won't have the luxury of talking to a server that's somewhere else. Computation and predictions that just have to happen on the edge, as opposed to happening in the infinitely scalable cloud, introduce limitations to the immediate availability of data and computing power.



Real-time machine learning can make it easier for edge devices because edge devices are constrained by memory and computing power. With batch predictions, processing and computation must happen in advance. In contrast, real-time systems generate predictions only for information that is relevant in the "now" which can actually save a lot of computation resources.

Finally, data coming in as real-time data is typically made available as batch data at a later time. So the more a system can push computation upstream, the fewer computations or data storage is required downstream. This works quite well for edge devices.

Q. What are some challenges that you have encountered while working with real-time ML models, and what did you do to rectify them?

Getting machine learning (ML) into production is challenging. In fact, it's possibly an order of magnitude more challenging than getting traditional software deployed. A fundamental blocker

preventing many teams from building and deploying ML to production at scale is that we have not yet successfully brought the practices of DevOps to machine learning.

MLOps solutions have emerged that solve the process of building and deploying ML models — but they lack support for one of the most challenging parts of ML: the data. In our post, "Why We Need DevOps for ML Data," we discuss why the industry needs to solve DevOps for ML data, how ML's unique data challenges stifle efforts to get ML operationalized and launched in production, and how Tecton aims to solve these problems.

Q. Are ML models trained in real time too?

Most that we've seen are not. But those that are, are basically online learning where a model running in production is continuously updated. These are relevant for use cases where the end user's behavior changes extremely quickly. However, in most use cases that we've come across, training ML models in real-time is not as impactful as giving real-time signals to a model that has maybe been trained once a day or once a week.

Q. Can you compare the value added by including real-time aggregation features as well as real-time non-aggregation features? For example, `num_orders_last_5_mins` vs `time_diff_since_last_payment_add`

It depends on the use case. Generally, real-time aggregation features are excellent for the types of features that are counting or summing something up or for a use case where the customer's behavior over the last couple of seconds or minutes is vital for an accurate prediction. For instance, for a fraud detection use case, a feature that counts how many transactions have happened on a credit card over the last 30 seconds could be substantial. If somebody is rapidly making lots of transactions with the same credit card across different merchants over 30 seconds, chances are that this user is behaving fraudulently. To make these predictions in the most timely manner, an aggregation feature here would be extremely important.

But in the case of a feature like the last known location of a user, aggregating or averaging the GPS coordinates isn't necessary. Real-time aggregation features are attractive when the aggregations happen over a time window; e.g., 30 seconds, a minute, or maybe even a lifetime.

Statistically, the variance and the feature value over the last couple milliseconds or seconds is a function of the size of the time window. Therefore, a lifetime transaction-counting feature's variance won't change. In real-time, using a real-time feature for lifetime aggregation features is typically not as

important. Oftentimes, batch lifetime aggregation features are sufficient unless these aggregations need to be made in a concise time window, in which case a real-time feature is necessary. Typically, the breaking point is somewhere on the order of a day.

Q. Can you compare the resources (price and efforts) difference between batch and real-time and recommend some less resource-intensive solutions for real-time?

The journey gets exponentially more complicated with real-time features. It's not uncommon for organizations to have teams of 5+ engineers dedicated to building the infrastructure to support real-time features. We built Tecton to help control associated costs and abstract the complexity of real-time systems. Tecton's core IP cost-effectively implements feature transformations and lets users control costs by tracking the computation and storage costs of individual features.

Q. Is there any use case that utilizes both batch ML and real-time ML simultaneously?

Many use cases use batch features, streaming features, and real-time features at the same time. Typically, however, they all feed into an online model designed to make real-time predictions.

Q. In step 2 (offline prediction served online + batch features), if the features are calculated offline, what's the gain of real-time inference? Wouldn't the model's predictions be the same as batch?

That is accurate. However, in some cases, pre-computing all the predictions requires too many entities to make it cost effective. By computing the model's prediction only at the time of the request, the overall cost of prediction can be controlled.

Q. Suppose our company's at stage 3 (online predictions + batch features). How hard would it be for a large company to transition to stage 4 (online predictions + batch and real-time features)?

It's possible. We've seen and helped many companies do it. First, ideally, the existing architecture is as modern as possible (i.e., where streaming infrastructure or access to raw data for on-demand features is already in place). Then, it's crucial to pick an obvious use case, a team that can move fast, and to secure executive buy-in. With executive buy-in and a well-defined use case where ROI is reasonably straightforward, it's much easier to show incremental wins that grow into additional use cases, additional teams, and, eventually, the enterprise as a whole. Start small, get a lot of buy-ins, and execute from there.

Q. How does Tecton backfill for non-aggregation streaming features, and how does it handle long-time window aggregations?

In Tecton, a Stream Feature View defines transformations against a Stream Data Source and can compute features in near-real-time. It processes raw data from a streaming source (e.g., Kafka or Kinesis) and can be backfilled from a batch source that contains a historical log of stream events. You can find out more here.

Long-time window aggregations present many challenges. For long-time window aggregations, the system needs to stitch together data from a stream and data from a batch source.

For example, in the case where a 30-day or 60-day type aggregation is necessary but a stream only contains data over the last two weeks, the system would need to break up the aggregations, use the previous 15 days' worth of data from the stream and then the 45 days' worth of data prior from the batch stores, and turn this into mini aggregations (i.e., daily counts, daily sums, etc.).

From there, at prediction time, the system would actually sum up these 60 daily counts to compute a new 60-day feature value. Users can leverage Tecton out of the box to combine batch data sources and stream data sources, bypassing one of the toughest challenges in feature design and processing.

Q. How does Tecton tackle the train-serve skew problem for non-aggregation streaming features?

With Tecton, end users express a streaming transformation in Python code, PySpark code, or Spark SQL code only once. That single streaming transformation will generate consistent training data and online feature values, all the while leveraging the same compute power underneath the hood. For example, suppose a user creates training data sets and online features with Spark. In that case, Tecton runs the exact same code for both operations, guaranteeing that the transformations are being run on the exact same compute engine with the same compute version, the same processing architecture, etc. This inherently reduces the risk of train-serve skew.

Q. As of now, Tecton only supports Spark streaming. Have you seen any performance issues with Spark streaming?

Projects like Flink come from a streaming background and added batch later, whereas Spark came from a batch background and added stream later. This probably explains why Spark Streaming isn't perfect, although Databricks, the company behind Spark, continuously invests in improving it. Spark Streaming has Spark Structured Streaming and Spark Continuous Streaming. Spark Structured Streaming does

stream processing in micro batches while Spark Continuous Streaming, as the name implies, continuously processes incoming data. Typically, Spark Continuous Streaming is faster and more cost efficient, but because it doesn't support all the same aggregations, like UDFs, it is more limited.

Q. Should on-demand transformations be saved to the offline store when they're sent to the online endpoint? Or should we log model requests and responses and take those on-demand features from there?

If logging is an option, teams should log as much as possible to reduce risks of train-serve skew and other problems. The big challenge with the logging approach is that it eliminates backfilling mechanisms. This isn't probablematic for super high-scale use cases where waiting just a few minutes for new logs results in enough new data to train a model. But with the vast majority of ML use cases out there, unless the team has the patience to wait weeks or months, backfilling mechanisms are necessary and very helpful.

Q. For low-latency serving and features that don't change frequently, is it better to send offline features to the online store only or to both the online and offline stores?

For low-latency serving, we definitely recommend leveraging a separate online and offline store. For batch features, you can synchronize the features from the offline store to the online store. For on-demand features and streaming features, you should avoid storing them in the offline store before they go to the online store. Instead, the features should execute online in real time.

Q. What is real-time machine learning, and why is it important?

The way we define real-time machine learning at Tecton is a machine learning application that is continuously running in the production system where there typically isn't a human in the loop. It's an application that constantly runs next to the other microservices that power a production application that is interfacing with an enterprise or a product's end users.

Q. What is an example of a use case you've come across in which switching from batch to real-time ML was the only way forward for a company not to hemorrhage money?

Fraud. Fraud. Fraud. Real-time machine learning is critical in fraud use cases. We define fraud as anything bucketed in with general abuse prevention where a malicious entity is trying to exploit a system in one way or another. This type of fraud exists in any company processing transactions, from organizations like Uber to banks and credit card companies. For example, by switching to real-time ML,

Instacart reduced the cost of fraud by a few million dollars a year. Batch for fraud only really makes sense in situations like wire transfers, where the transaction takes 24 hours or longer to be processed.

Q. What are the most common use cases for real-time machine learning?

Typical examples are fraud prevention, recommendation systems, ranking and search, dynamic pricing, and personalization. Insurance companies and banks use real-time machine learning to assess the risk associated with a customer to determine whether they want to approve a loan.

Another example of a consumer-facing real-time ML application is UberEats: The inputs that feed into the model come from various batch, real-time, and streaming data sources. Data includes how long an order is going to take, how many deliveries are currently out on the road, the current and future location of all the drivers, the fastest way for a driver to go by the restaurant before heading to the customer's delivery address, how many purchases or orders a particular restaurant has been able to process in parallel in the past correlated with how many orders the restaurant is currently processing as a proxy for wait time, and so on.

All of these raw signals need to be turned into machine-learning features, which are, essentially, highly curated, nicely formatted, cleaned-up data points that a machine-learning model is trained on before it can actually come up with a final prediction in production. In the case of Uber Eats, different machine learning models use different features from raw inputs to predict a delivery time or to recommend a top restaurant to a customer.

Q. What's the percentage of enterprises already using real-time machine learning?

In one user study conducted by Chip, she found that only about 5% of the people she spoke to wanted to use real-time features. But when the organizations pushed to get the appropriate infrastructure in place, they found that by moving to real-time ML, some of their use cases significantly increased performance, user experience, and revenue.

Real-time ML is also very use-case dependent. For instance, almost 100% of transaction processing credit card companies already use real-time machine learning in one way or another. But getting the proper infrastructure in place is crucial as well as challenging. So in the grand scheme of things, real-time machine learning is still in the super early innings of becoming mainstream for most companies.

Q. The perception today is that moving to real-time ML requires significant infrastructure lift. How does an organization know, before committing to the infrastructure lift necessary for real-time, that this investment will have a positive return?

Ideally, an organization should perform back testing as much as possible. If a model is not fully running in production yet, the organization can train it offline and assume that they've replicated offline the same outcomes that online real-time data would yield.

For instance, as part of the process of generating a training data set and testing the features made available to the model, a data scientist or ML engineer could decide the level of freshness of the features they are serving to the model. For example, they could choose to hide or include features that are a minute, an hour, 12 hours, or a month old. If done rigorously, this process would allow them to map out how the model performs depending on these time-windowed features and help them asses if introducing real-time data could create significant uplift.

Of course, this wouldn't account for the whole picture because ML applications are often subject to second-order effects where a prediction could actually impact a user's behavior. For example, in the case of Lyft or Uber, testing dynamic pricing with a high level of confidence is nearly impossible due to the uncertainty of how users and drivers will react to price changes. In this situation, it makes more sense to run an experiment in production, perhaps as an MVP at a limited scale, to assess how users actually react and how the prediction impacts their behavior and to include those results in future predictions on additional users.

Q. What are some of the most common challenges of building features from raw data sources?

Data Source: One challenge when it comes to building features from raw data sources is that depending on the data source—whether it's a data warehouse, a transactional data source, a stream source, or in-memory data like prediction request data—the characteristics of that data vary enormously. Data from a data warehouse typically contains the complete history and can enable large-scale batch aggregations. In contrast, data from a stream may typically only include information from the last two weeks and only allow time widow aggregations or very recent role-level type transformations. This constrains the types of features that data can yield.

Latency: The next challenge comes down to serving those features in production. The application making the prediction will not be able to ship every query against a data warehouse instantaneously or

in a matter of milliseconds. That's where an online feature store or production store between the model and the data warehouse comes into play. This setup decouples feature calculation from feature consumption so that the predictive application can fetch feature values at the serving latency that the model requires.

Train-serve skew: Even when all the different infrastructure pieces are stitched and wired up together, one of the biggest challenges is identifying and debugging training-serving skew. Train-serve skew happens when models are trained on features that would never exist in production. For example, let's take a system with two different feature implementations, offline and online. Suppose those implementations contain even the most minor differences, like amputating a null value slightly differently. In that case, the model will make predictions in the production system that don't follow the same rules as the testing environment. In the best-case scenario, the predictions will be horribly wrong and, therefore, easily detected. In the worst case, the predictions are just not as good as they could be, and the problem can go undetected or misunderstood, resulting in important losses over time.

Monitoring: Finally, monitoring feature values can also be very tricky. These monitoring challenges are, in a way, related to the risk of train-serve skew because a well-implemented monitoring system will make sure that the features the stores are serving are valid, that they fit a specific schema range, and that they are consistent online and offline.

Q. What are the most common architectures you've seen for all the stages you describe?

Batch ML (offline predictions served online and batch features) – stage 2: In stage 2, the system is making predictions offline, and storing and serving those predictions online. In a simple architecture that leverages standard open-source technologies, the setup could look something like this:

1. A warehouse or datalake stores raw data,
2. a scheduler like Airflow or Prefect orchestrates a job that processes raw data and transforms it into ML features,
3. another job executes batch inference with an ML system like Spark or Acute Flow,
4. the system then stores predictions in an online store like DynamoDB or Redis.
5. From there, a real-time API in front of the online store serves predictions online.

Online prediction with batch features – stage 3: In this scenario, the system only leverages batch features that some ETL process or solution like Tecton helps store synchronically across an offline and

online store. This architecture requires a machine learning model running in a microservice or in a Kubernetes pod that fetches the features in real time from the online store and then serves the prediction to an end customer or to an application in real time.

Online predictions with real-time features – stage 4: Whereas the batch feature setup in this architecture remains simple, streaming transformations make these systems much more complex to build, orchestrate, and maintain. Streaming feature pipelines are continuously processing data from streaming infrastructure like Kafka or Kinesis and making the calculated output available as a key-value store in the online store. Separately, the streaming features must exist in the offline store, and the system must be able to backfill features for any new features that come up to train models.

On-demand transformations: These transformations process data on demand (meaning at the time of prediction) and typically leverage data only available in memory, in some microservice, or in an online database. Examples of these on-demand transformations range from an end user's GPS location that yields a Geohash or an IP address that yields a county, etc. These transformations are typically in the hot path of real-time predictive applications and must, therefore also find a way into offline stores for training. A common way to solve this challenge is feature logging where the system logs the production outputs. These logs can then help train models down the road. Online inference brings all these different jobs and paths together for accurate predictions in production.

Q. Real-time ML systems typically use more straightforward, less complex models than batch processing systems to achieve real-time performance. Can this limit the system's ability to make complex predictions or handle complex data?

In general, batch processing systems have the luxury of time, which real-time processing systems don't have. That's why you see strategies employed in real-time processing systems like:

- Pre-computing expensive feature computations in batch and serving them online
- Horizontal + vertical scaling
- Use-case optimized hardware (like GPUs)

Those strategies, of course, have their limits, but the industry has come quite far in pushing those limits—just think about the complexity of the real-time data processing that self-driving cars have managed to overcome.

Q. Both Twitter and Uber tried to go from Lambda to Kappa (pure real-time) but failed; seems it's tough to go pure real-time. What are your thoughts on this?

This topic requires a dedicated podcast, book, or more. However, a few quick thoughts: At the end of the day, batch and stream processing are very similar. Streams are just an infinite batch of data. Technical challenges have prevented the industry from using them exactly the same (dealing with late-arriving data, optimizing the data layout for specific query patterns, etc.). However, industry players keep pushing the envelope here to bring batch and stream processing so close to each other that the end user doesn't even have to think about their difference anymore and can use the same technology for both. This explains why today, we're still seeing more Lambda-based architectures than Kappa-based ones

Q. What are the most significant risks with real-time ML regarding reliability? What can be done to ensure real-time ML works reliably?

Reliability is essential and introduces much room for issues with data quality, data drift, pipeline health, online serving latencies, uptime, etc. If an organization builds its own stack, they have to set up the team and infrastructure to support this themselves. It's tricky and usually very costly in engineering resources. We've designed Tecton from the ground up to handle this for its users and serve features in production at extreme scale for batch, streaming, or real-time ML with the confidence that systems will always be up and running. To find out more about how Tecton's best-in-class cloud services maximize resilience and scale, please click here.