Real-Time Machine Learning

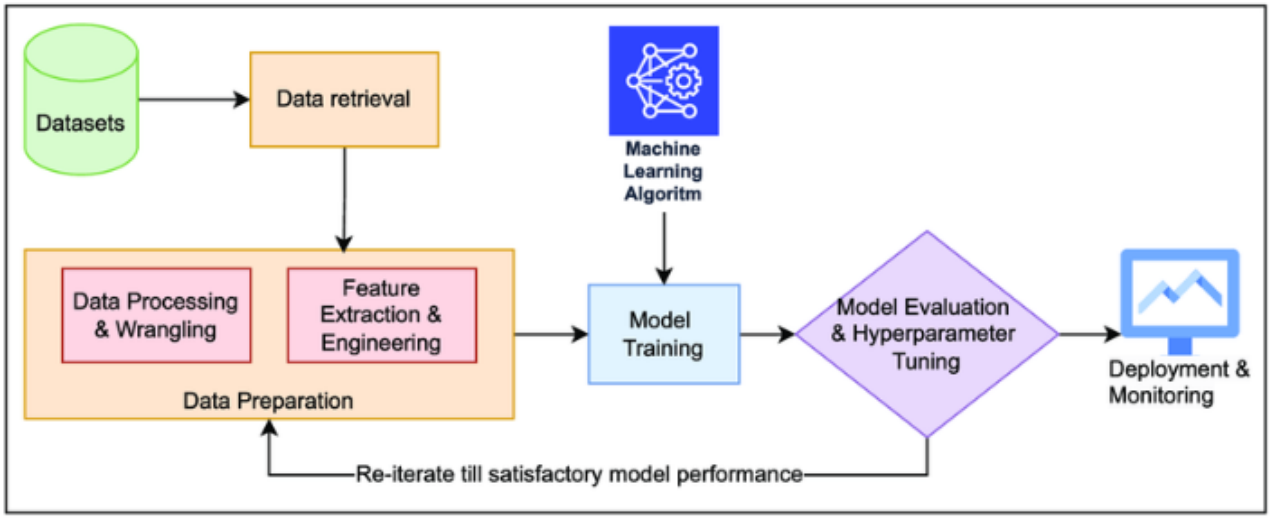
Real-time machine learning (ML) refers to the practice of making predictions, decisions, and learning from data as events occur, often within milliseconds to seconds. This paradigm is distinct from traditional batch ML systems, which retrain and predict on fixed intervals or historical data. As applications like fraud detection, recommendation systems, dynamic pricing, and chatbots demand immediate responses, real-time ML has become increasingly critical.

---

1. ML Pipeline Overview

A modern machine learning pipeline includes several key stages:

- **Data Collection**: Ingesting data from logs, event streams, APIs, etc.
- **Data Storage**: Using data lakes (e.g., Amazon S3) and data warehouses (e.g., BigQuery).
- **Feature Engineering**: Using tools like Feast to transform raw data into meaningful features.
- **Model Training**: Training models in batch or continually.
- **Prediction Serving**: Deploying models for batch or online inference.
- **Monitoring and Feedback**: Evaluating predictions and feeding data back into the loop.



---

2. Batch Prediction, Online Prediction and Batch Prediction with Online Capabilities(Hybrid)

2.1 Batch Prediction

- Batch prediction refers to generating predictions for many data points at once, typically on a scheduled basis
- Prediction are computed and stored and inference happens offline(ahead of time). It isn't responsive to real time data. Often run as ELT vs ETL jobs.
- Its use cases include spam classification, custom churn scoring, Credit risk scoring, etc.
- Batch Prediction typically runs on Data warehouses or Data Lakes. Output is stored in form of KV Pairs. Computation is done using Spark or Python pipelines.
- It is cost efficient, easy to scale.
- Major disadvantage is predictions become stale and it isn't real time. This can lead to a lot of train-predict inconsistency.

2.2 Online Prediction

- Real-time predictions on demand. Inference happens per request. Input features may come from real time input streams or precomputed stores. It requires infrastructure like RESTAPIs.
- Required for time-sensitive systems like fraud detection, chatbots, autonomous driving systems, etc.
- Needs low-latency and high frequency infrastructure, feature lookup in KV stores like Redis, DynamoDB.
- It is more adaptive to concept drift.
- Major con is Risk of feature computation lag or missing values. Need careful monitoring of latency and throughput

2.3 Batch Prediction with Online Capabilities

- It uses batch prediction as defaults, but refine or adjusts with real time inferences.
- It ofen uses a layered architecture, where in layer 1 had precomputed results in a KV store, while layer 2 is a lightweight model.
- Used in e-commerce recommendations, news apps, etc.
- It applies re-ranking and last minute personalization, something like the instagram algorithm.

---

3. Data Storage: Data Lake, Data Warehouse, and Key-Value Stores

- **Data Lake**: Stores raw, unstructured/semi-structured data. Good for flexible access and cost-effective archiving.
- **Data Warehouse**: Optimized for structured data analysis and reporting.
- **Key-Value Store (e.g., DynamoDB)**: Enables fast feature lookups during real-time inference.

---

4. Near Real-Time Communication

- Achieved via message queues and stream processors like Kafka, Flink, Redis Streams.
- Enables fast ingestion and propagation of event data.
- Crucial for updating models or triggering inference pipelines in response to user actions.

---

5. Train-Predict Inconsistency

Train-predict inconsistency occurs when the features or data used during training are different from those used during inference (prediction). This inconsistency leads to a drop in model performance when deployed, despite good results in offline evaluation.

5.1 Why it can happen?

- Feature transformations are implemented separately for training and for inference. Ex) Feature transformation implemented in pandas, but inferred in Java or Go.
- Logic duplication leads to subtle mismatches. Ex) Using last 7 days of activity in training but calculating it in 5 days over production.
- Temporal mis-alignment like features use data in training that wouldn't be available in inference.
- Data Leakage and Feature Drift.

5.2 How to prevent this from happening?

1. Using Unified Feature stores like Feast or Tecton that support both real time band batch inference.
2. Version and Log feature pipelines.
3. Temporal Validation
4. Use monitoring tools to avoid drift.

---

6. Concept Drift and Model Drift

Concept Drift refers to the phenomena where the statistical relationship between input features X and the target variable Y changes overtime. In other words, "The model you trained doesn't understand the data anymore".

6.1 Types of Drift:

1. **Sudden or Abrupt Drift:** Concept changes drastically in a short amount of time. Ex) A new regulation causes a sudden drop in loan approvals.
2. **Gradual Drift:** The change happens slowly over time. Ex) Customer preferences for certain products shifts slowly over a season.
3. **Incremental Drift:** Decision boundary itself moves. Ex) Average click through rate on ads decreases as users get used to banner ads.
4. **Recurring Drift:** Pattern re-occurs periodically.

6.2 Detecting Concept Drift:

1. **Performance Monitoring:** Using metrics like F1, precision, etc over time. Only downside to this method is, it requires labelled data.
2. **Distribution Monitoring:** Using tests like KS test, Population stability, Jensen Shannon Distance

Concept Drift

---

7. ML Training Strategies

1. **Continual Learning:** It is also known as Online Learning. The model learns from a stream of data points. Weights are updated incrementally after each sample or mini-batch. It is

common in environments requiring low-latency requirements and fast-changing data. Techniques include online gradient descent, rehearsal buffers. It prevents Stale models.

2. **Stateless Learning:** Retrain a new model from scratch using most recent data. No memory of previous models.
3. **Cadence Based Training:** Retraining happens at regular scheduled intervals.
4. **Stateful training:** - The model maintains internal state or context between training cycles. May use rolling memory of past examples, parameter history, or learned features. Ex) LSTMs/Transformers trained with sequential patterns in time series. Learns temporal patterns. Useful in time-dependent or session-based modeling.
5. **On Demand Training:** The model is retrained reactively — only when needed. Triggers: data drift, concept drift, performance degradation, or business rule violation. ex) Credit scoring model retrained only when accuracy falls below 90% or if PSI > 0.2. Saves compute and avoids unnecessary retraining. Responds to real business signals or failures.

7.1 On Demand Stateful Training

- This approach combines- Drift- or event-triggered retraining (on-demand) AND Preserving model knowledge from the previous iteration
- The system monitors for concept drift or degradation in performance. When an alert is triggered, it loads the latest model checkpoint and continues training using the new data. The model evolves only when necessary and doesn't "reset" itself.

7.2 Stateful Cadence Based Retraining

- This approach combines Regular (cadence-based) retraining AND Preserving model knowledge from the previous iteration.
- The model is retrained on a fixed schedule (e.g., daily or weekly), but instead of restarting from scratch, it begins from the latest saved model checkpoint and continues learning from newly arrived data. Previous weights, embeddings, or memory representations are retained and updated rather than discarded — enabling the model to accumulate knowledge over time while staying up-to-date.

---

8. A/B Testing in Real-Time ML

A/B testing is a method used to compare two or more versions of a product, feature, or machine learning model to determine which one performs better based on defined metrics. Group A is called control group and Group B is called test group. Group A is tested upon model v1 and Group B is trained upon model v2.

A/B testing helps to validate whether a new model or feature actually improves the performance or not. It helps to make data drive decisions before deployement.

8.1 How it works

- Model A (v1) continues serving most users.
- Model B (v2) handles a small percentage of users.
- You track precision, recall, F1, fraud capture rate, and false positive rate separately for both.

---

9. Engineering Challenges in Real-Time ML

- Low latency requirements (sub-100ms responses).
- High throughput environments (e.g., ad platforms).
- Managing streaming data ingestion.
- Ensuring feature freshness.
- Consistency between training and inference.
- Monitoring for failures and performance degradation.

---

10 Case Studies:

Michaelangelo and DataBricks are the quintessential case studies for RTML.

10.1 Michael Angelo

Michael Angelo by Uber was the first fully functional RTML project that was deployed.
Official Michaelangelo Blog

10.1.1 Features of Michaelangelo

Component	Description
Online Feature Store	Provides real-time and batch features consistently across train + serve.
Real-Time Inference	Supports low-latency model serving for fraud detection, ETA prediction, etc.
Streaming Ingestion	Uses Kafka and Flink for real-time event ingestion and feature computation.
Model Deployment	Deploy models via REST APIs (real-time), or schedule batch scoring jobs.
Model Monitoring	Tracks drift, feature quality, latency, and prediction accuracy.

10.1.2 Operation carried out by Michaelangelo

- Driver ETA predictions
- Real time focused fraud detection for Uber Eat or Rides payments.
- Dynamic Pricing depending upon the market conditions.
- Food delivery logistics optimizations.

10.2 DataBricks

Databricks is a cloud-based platform that serves as a one-stop shop for all data needs, such as storage and analysis. It was created by the people behind Apache Spark. Databricks can generate insights with SparkSQL, link to visualization tools like Power BI, Qlikview, and Tableau, and develop predictive models with SparkML. You can also use Databricks to generate tangible interactive displays, text, and code. One could say it’s a MapReduce system alternative.

Databricks deciphers the complexity of data processing for data scientists and engineers, allowing them to build machine learning applications in Apache Spark using R, Scala, Python, or SQL interfaces.

10.2.1 Features of DataBricks

Component	Description
Delta Live Tables (DLT)	Supports continuous ingestion and transformation of streaming data.
Feature Store	Centralized storage for features — supports both offline and online lookup.

Component	Description
MLflow	Manages experiments, model tracking, versioning, and deployment.
Model Serving (Native)	Deploys models as REST endpoints — supports real-time predictions.
Streaming + Batch	Supports hybrid pipelines: streaming ingestion → batch model retraining.

10.2.2 Operations carried out by DataBricks

- Delta Lake ensures ACID() guarantees for streaming data.
- Demand forecasting using streaming sales data.
- Recommendation systems personalized per user session.
- Predictive maintenance for IoT sensor streams

10.3 Patterns across both / Conclusion:

Both platforms implement the Real-Time ML loop:

1. Streaming data flows into a feature store (Kafka/Flink or Delta Live Tables).
2. Features are shared across training and prediction for consistency.
3. Models are trained regularly using batch jobs or online learning.
4. Real-time predictions are served via low-latency REST/gRPC APIs.
5. Monitoring detects drift, triggering retraining if needed.

---

11. Future Directions

- Edge ML for ultra-fast inference on-device.
- Federated learning to train models across private user devices.
- Streaming AutoML pipelines.
- Integration of small, efficient LLMs in real-time tasks.
