Real-Time Machine Learning Systems

Jim Dowling
jdowling@kth.se

Real-Time and Interactive Systems with SLOs

- A real-time system processes messages in a bounded amount of time.
- The upper bound on the time available to process the request or event is defines how “real-time” by the system requirements.
- Due to the best-effort nature of the Internet Protocol, Internet Services provide service level objectives (SLOs) for the maximum latency for processing data.
- Interactive (user-facing) systems have SLOs - they need to return results before users decide that the service is too slow and stops using it.

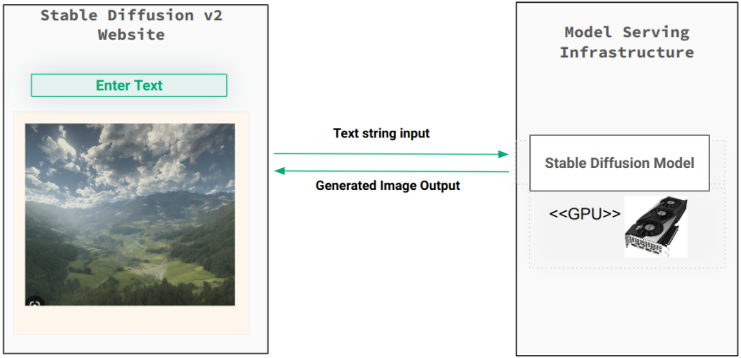
[Image from http://retis.sssup.it/~giorgio/courses/rts/rts.html]

Higher latency reduces usage of Interactive Systems

	Distinct Queries/User	Query Refinement	Revenue/User	Any Clicks	Satisfaction	Time to Click (increase in ms)
50ms	-	-	-	-	-	-
200ms	-	-	-	-0.3%	-0.4%	500
500ms	-	-0.6%	-1.2%	-1.0%	-0.9%	1200
1000ms	-0.7%	-0.9%	-2.8%	-1.9%	-1.6%	1900
2000ms	-1.8%	-2.1%	-4.3%	-4.4%	-3.8%	3100

Bing's test: Bing delayed server response by ranges from 50ms to 2000ms for their control group. You can see the results of the tests above. Though the number may seem small it's actually large shifts in usage and when applied over millions can be very significant to usage and revenue. The results of the test were so clear that they ended it earlier than originally planned. The metric Time To Click is quite interesting. Notice that as the delays get longer the Time To Click increases at a more extreme rate (1000ms increases by 1900ms). The theory is that the user gets distracted and unengaged in the page. In other words, they've lost the user's full attention and have to get it back.

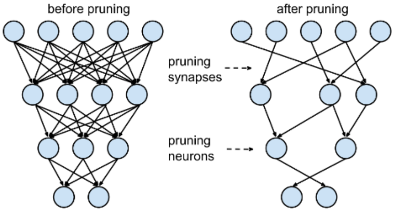
Example: Hugging Face Spaces with Stable Diffusion



Note 1: Stable Diffusion v2 does not use any history or context info - only the user input is needed to generate the prediction.
Note 2: if we replaced Stable Diffusion v2 with ChatGPT, we would have the same simple stateless model serving system.

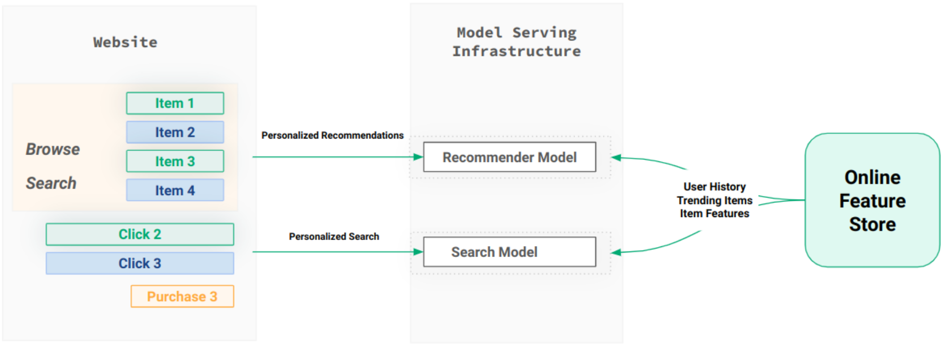
Reduce Model Inference Latency with Distillation

- Big models, such as Stable Diffusion, have high latency (seconds) even on GPUs, such as the Nvidia A100.
  - "On Distillation of Guided Diffusion Models" reduces the latency of Stable Diffusion by a factor of 10 through progressive distillation.
- The Lottery Ticket Hypothesis (Frankle and Carbin) identifies situations where a smaller network can be trained to a similar accuracy as a large network.



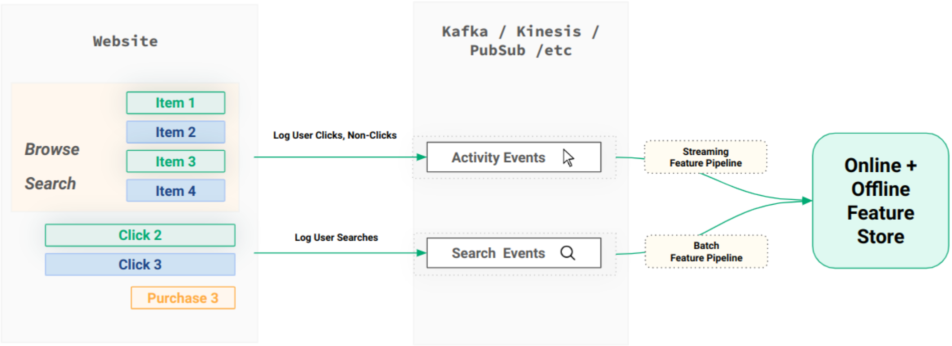
Model Distillation prunes the number of weights and amount of compute needed for inference, while minimally affecting model performance
Image from https://herbiebradley.com/The-Lottery-Ticket-Hypothesis

Personalized Models require History and Context (Feature Store)



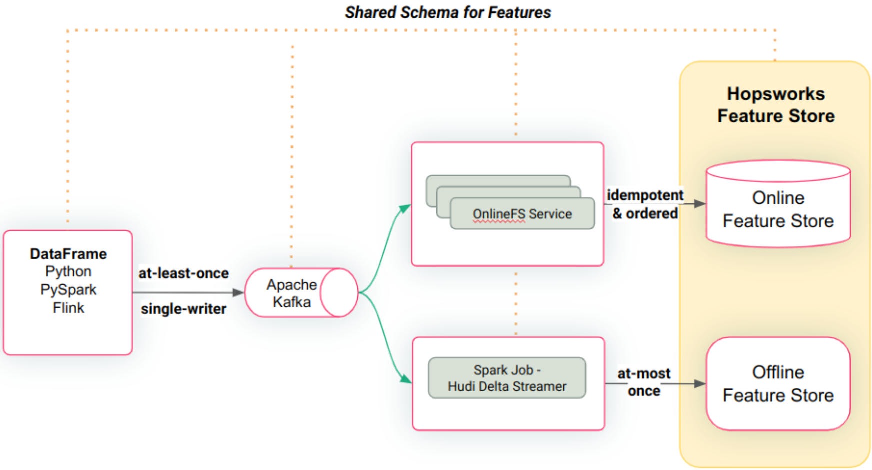
Note: If our model requires history or state (for users, in this example), then we need to plug in a feature store to provide the precomputed historical and contextual feature values.

Feature pipelines write to both the Online/Offline Feature Stores



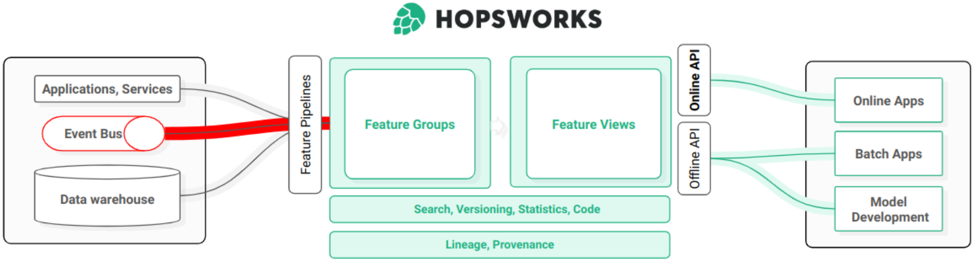
You need to instrument your retail website to generate the events used to compute the historical/contextual features. Some features are computed by batch programs, but some features are fresh and computed with streaming pipelines. The **features can be stored in both the offline and online stores of the feature store**.

When writing, Ensure Consistency Between Offline and Online Feature Stores

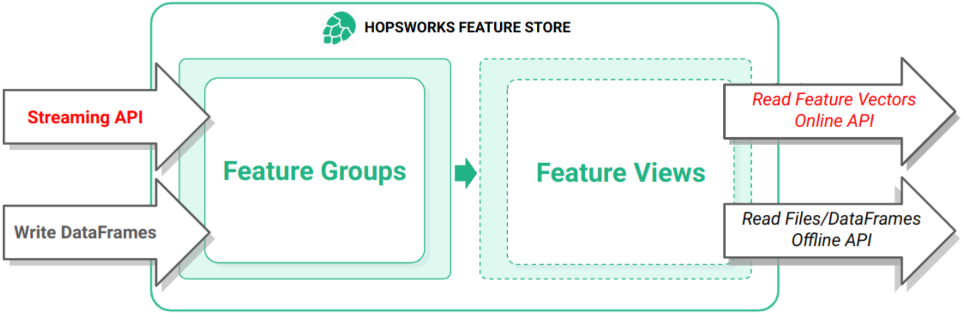


The Online Feature Store

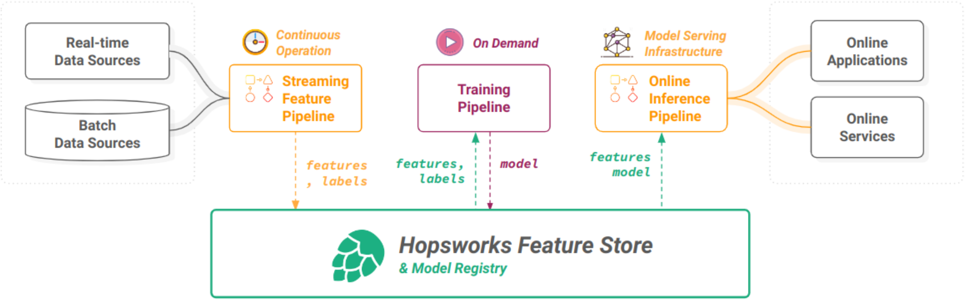
Hopsworks: Write to Feature Groups, Read from Feature Views



Real-Time APIs for Writing and Reading to the Feature Store



Streaming/Training/Online-Inference Pipelines



Programming Frameworks for Streaming Feature Pipelines

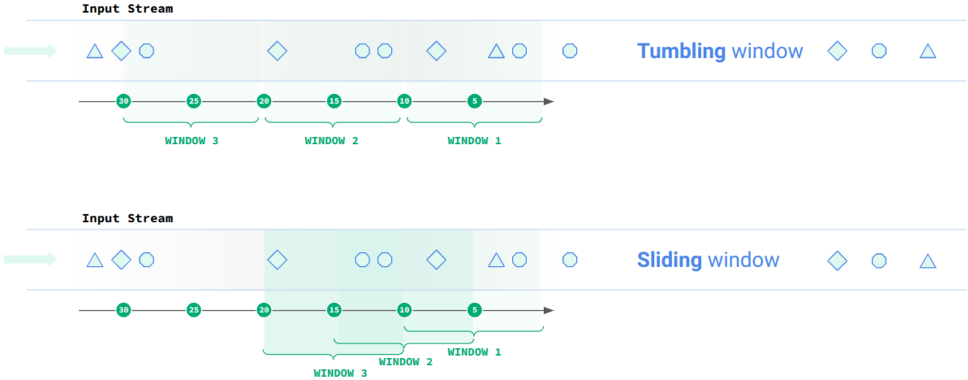
Real-Time Features using the Hopsworks' Streaming API

- Apache Spark Streaming
- Apache Flink

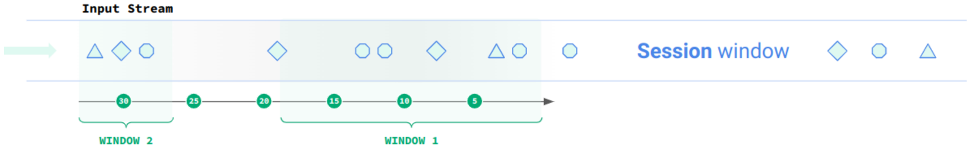
A streaming framework in Python

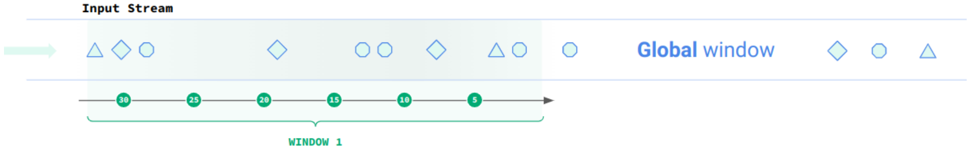
- Bytewax (Uses Kafka for Stateful Recovery)

Streaming Feature Pipelines - Windows (1/2)



Streaming Feature Pipelines - Windows (2/2)

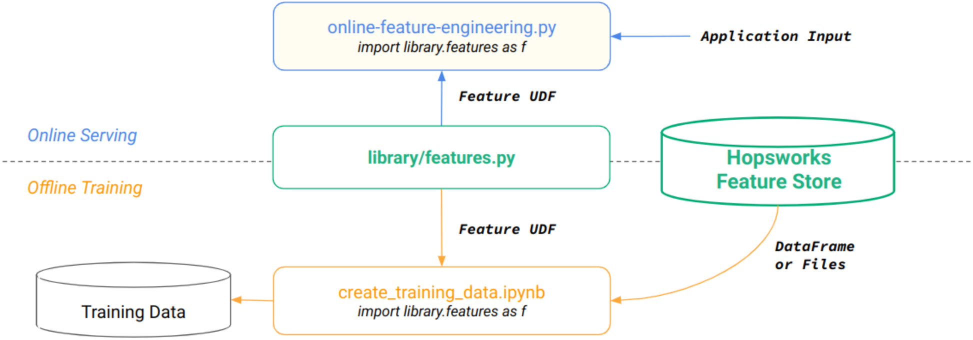




PySpark Streaming Program

```
df_read = spark.readStream.format("kafka")...option("subscribe",
KAFKA_TOPIC_NAME).load()

# Deserialise data from Kafka and create streaming query
df_deser = df_read.selectExpr(....).select(...)

# 10 minute window
windowed10mSignalDF = df_deser \
    .selectExpr(...)\
    .withWatermark(...) \
    .groupBy(window("datetime", "10 minutes"), "cc_num").agg(avg("amount")) \
    .select(...)

card_transactions_10m_agg =fs.get_feature_group("card_transactions_10m_agg", version = 1)

query_10m = card_transactions_10m_agg.insert_stream(windowed10mSignalDF)
```

On-Demand Features



On-Demand Features - Example

UDF (user-defined function) for computing an on-demand feature (haversine distance)

```python
def haversine_distance(long: float, lat: float, prev_long: float, prev_lat: float)-> float:

        … check/cast parameters as a Pandas Series (batch API for training)
        … or keep as primitive values (online inference)

        prev_lat = radians(prev_lat)
        long_diff = prev_long - long
        lat_diff = prev_lat - lat

        a = np.sin(lat_diff/2.0)**2
        b = np.cos(lat) * np.cos(prev_lat) * np.sin(long_diff/2.0)**2
        c = 2*np.arcsin(np.sqrt(a + b))

        return c
```

Online or Interactive Applications

Making an Application "Intelligent" with a Model and Features

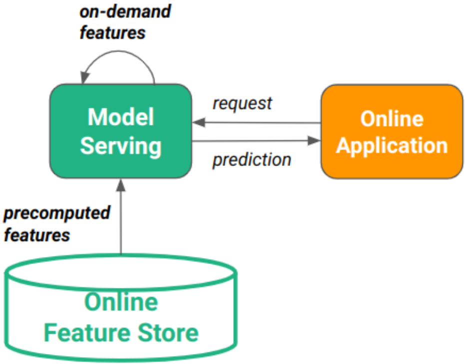
- Most interactive applications are stateless
  - Separating storage and compute makes it easier to build highly available systems. State is stored in an operational database and/or object store.

- Interactive applications need a model and sometimes a feature store
  - Host the model in model serving infrastructure to decouple it from the app

- Use a feature store if the application needs
  - historical information to make a decision
  - context information in the system to make a decision

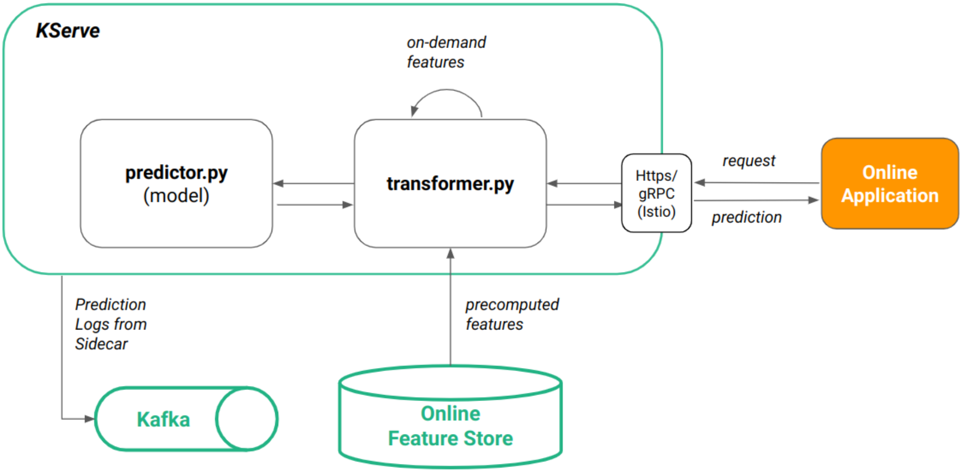
- Features can be
  - computed on-demand if the feature is based on live input data
  - precomputed as historical features using a feature pipeline
  - precomputed as contextual features using a feature pipeline

Online Inference and the Online Feature Store



The Online Feature Store provides low-latency, row-oriented access to features. You provide the primary keys, and it returns feature vectors.

KServe - Open-Source Model Serving Infrastructure



Example transformer.py program

```
class Transformer(object):
    def __init__(self):
        project = hopsworks.login()
        fs = project.get_feature_store()
        self.trans_fv = self.fs.get_feature_view("transactions", 1)
```

get reference to feature_view

```
    def preprocess(self, inputs):
        cc_num = inputs["inputs"][0]["credit_card_number"]
        feature_vector = self.trans_fv.get_feature_vector({"cc_num": cc_num})
        # compute any on-demand features needed here
        return { "inputs" : [{"features": feature_vector.values.tolist()}] }
```

get feature vector for cc_num from feature store

```
    def postprocess(self, outputs):
        preds = outputs["predictions"]
        return preds
```

any changes needed before returning the prediction

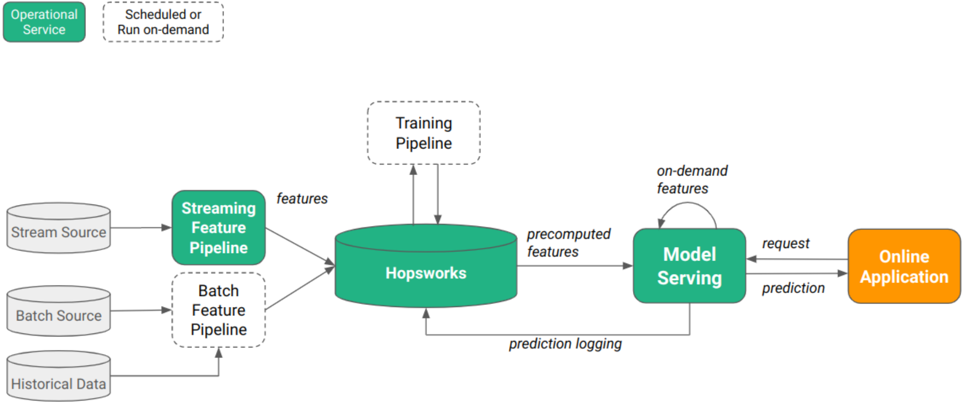
Example predictor.py program

```python
import joblib
import numpy as np

class Predict(object):
    def __init__(self):
        self.model = joblib.load("/path/to/model.pkl")

    def predict(self, inputs):
        features = inputs[0]
        return self.model.predict_proba(features).tolist()
```

load model

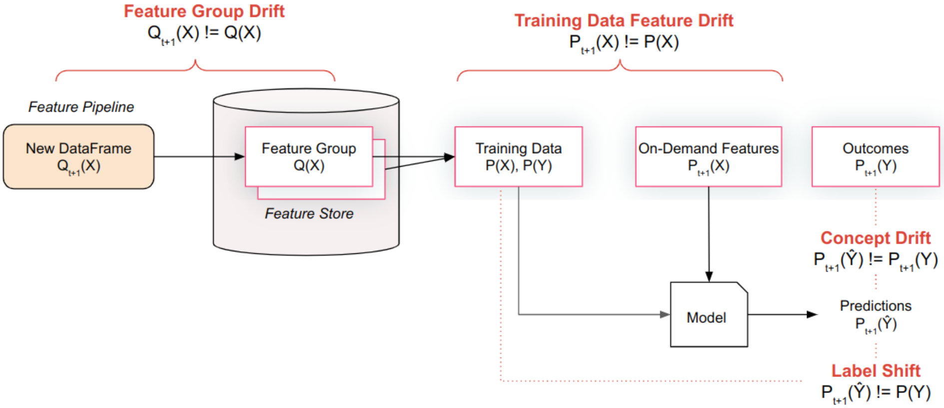
make prediction with model

Putting it together in an Operational ML System

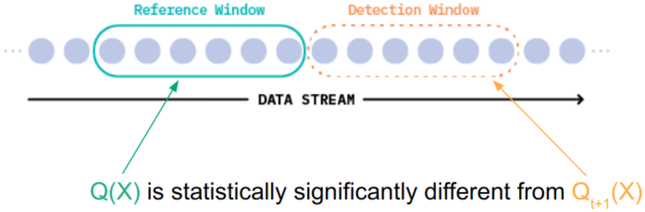


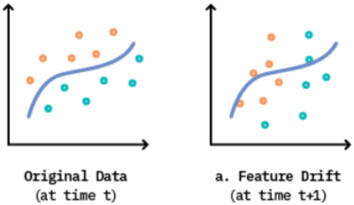
Online Feature/Prediction Monitoring

What should you monitor in a ML System?



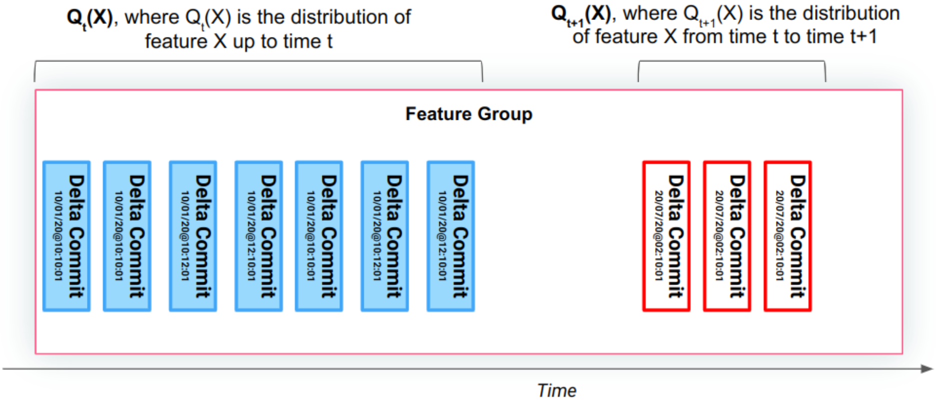
Feature Drift





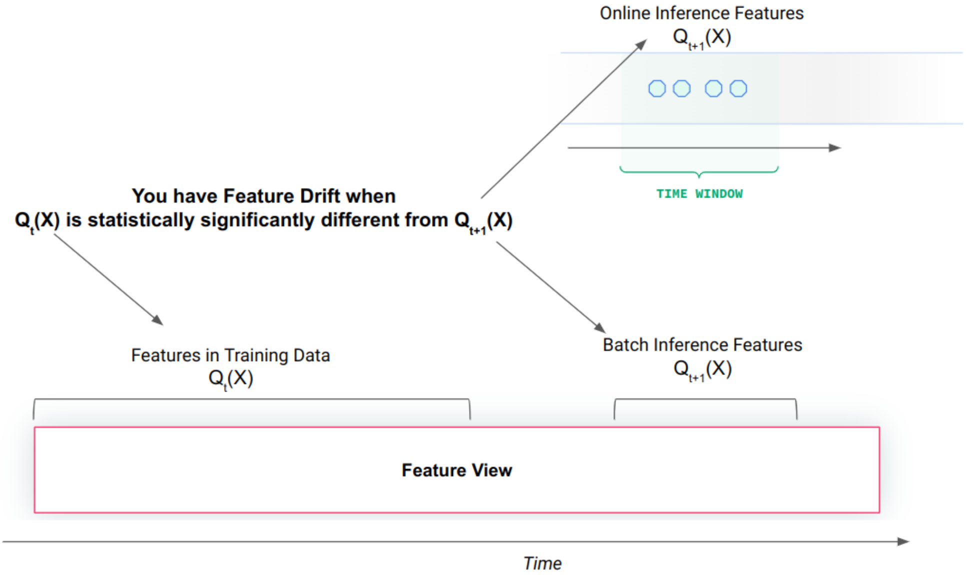
Images from [https://concept-drift.fastforwardlabs.com/]

Feature Group Drift



You have Feature Drift when
$Q_{t+1}(X)$ is statistically significantly different from $Q_t(X)$

Training Data Feature Shift



Models Degrade in Quality over Time

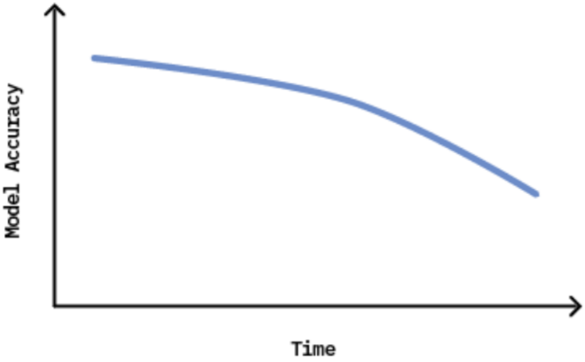
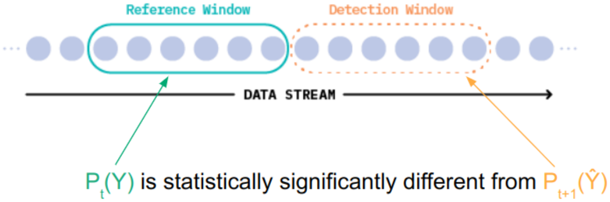


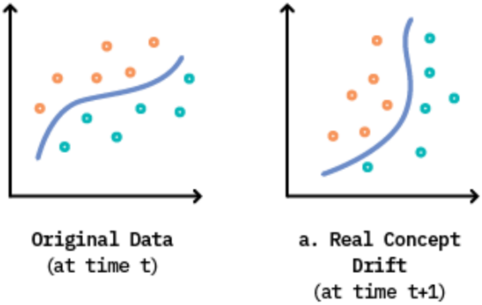
Image from https://concept-drift.fastforwardlabs.com/

Label Shift



For example, in the training data for our credit card fraud detection dataset, 0.001% of the credit card transactions are labelled as fraud.
In inference, however, 0.003% of the credit card transactions are predicted to be fraud.
Is this because there is more fraud is actually happening or because the model is degrading in quality?

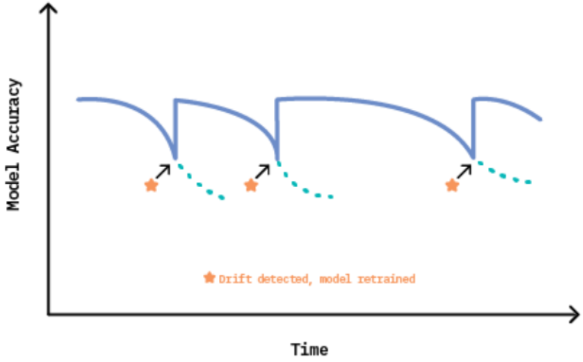
Concept Drift - where the model degrades over time



$$\text{accuracy}(P_{t+1}(Y), P_{t+1}(\hat{Y})) < \text{accuracy}(P_t(Y), P_t(\hat{Y}))$$

Image from https://concept-drift.fastforwardlabs.com/

How to handle Concept Drift - Retrain models



Here, a model is retrained after drift is detected, and it's accuracy improves, only to decay over time, requiring periodic retraining to keep model accuracy high.

Image from [https://concept-drift.fastforwardlabs.com/]

Algorithm for Detecting and Retraining models

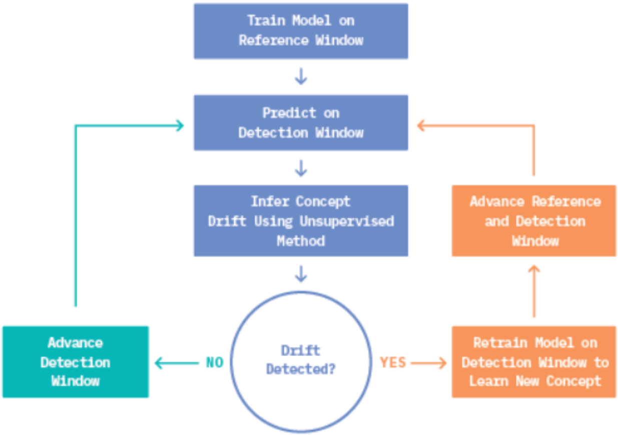


Image from [https://concept-drift.fastforwardlabs.com/]

TikTok - online model retraining for recommendations

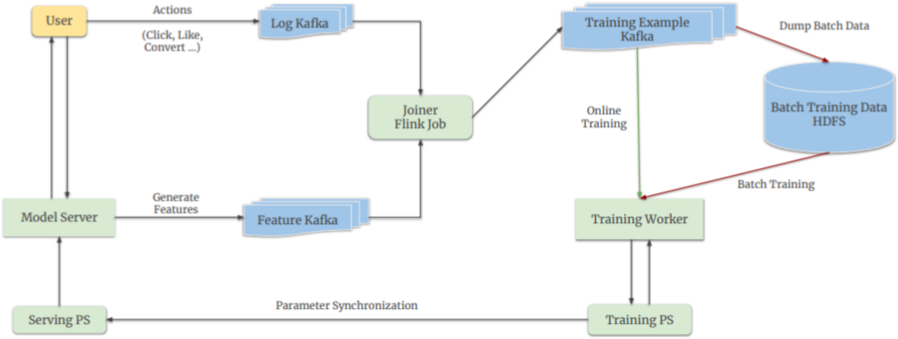
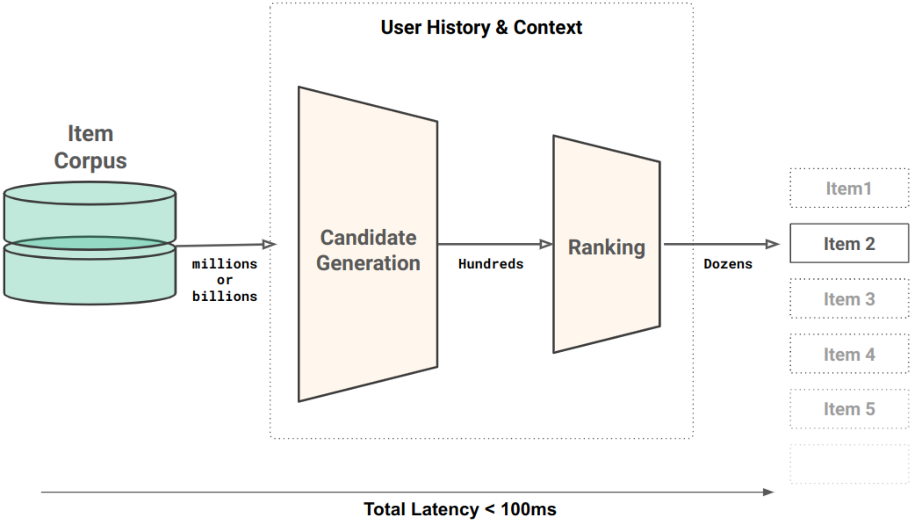


Figure 4: Streaming Engine.
The information feedback loop from [User → Model Server → Training Worker → Model Server → User] would spend a long time when taking the Batch Training path, while the Online Training will close the loop more instantly.

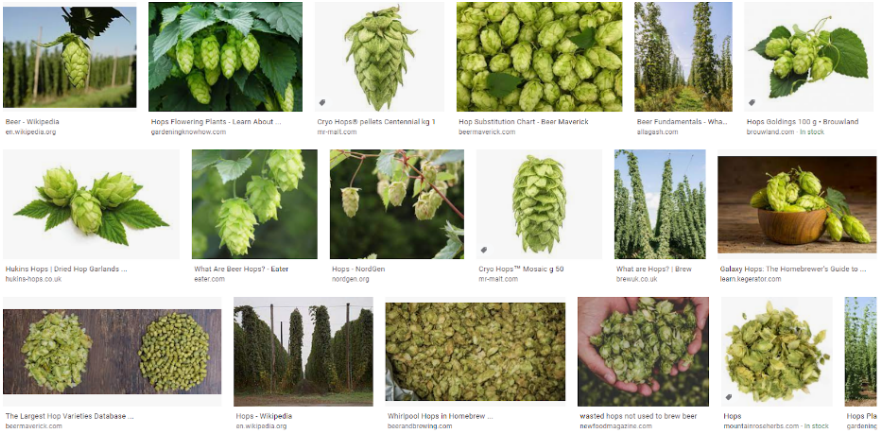
https://arxiv.org/pdf/2209.07663.pdf

Real-Time Personalized Recommendations/Search

Real-Time Personalized Recommender/Search Service



Embeddings can be used for Similarity Search with a VectorDB



"Find me a similar image" - Similarity Search using Image Embeddings (Google)

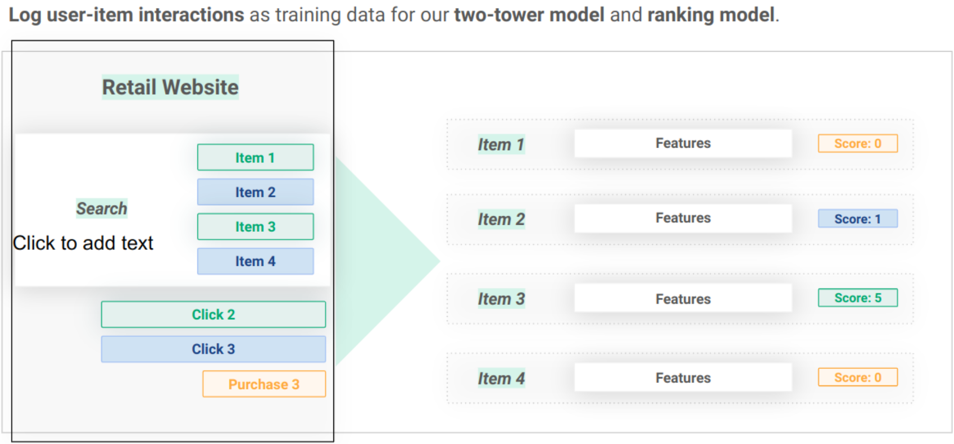
What about Multi-Modal Similarity Search?

Can a “`user query`” find “`items`” with similarity search?

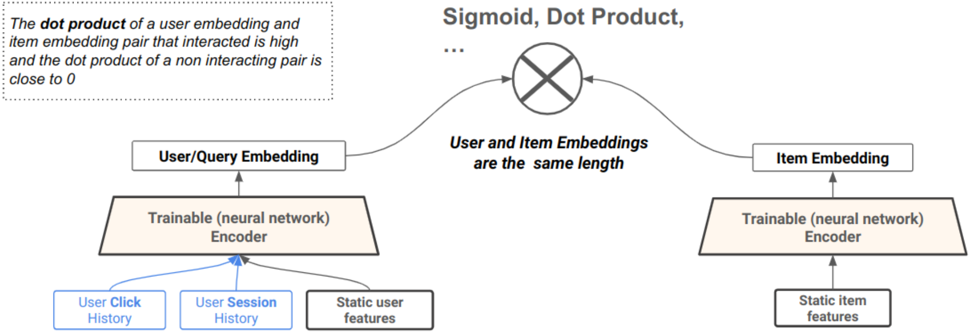
Yes, by mapping the “`user query`” embedding into the “`item`” embedding space with a **two-tower model**.

*Representation learning for retrieval usually involves supervised learning with labeled or pseudo-labeled data from **user-item interactions**.*

Training data for our Two-Tower Model will be User-Item Interactions



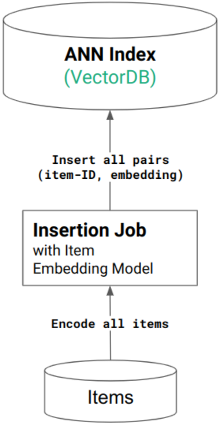
Two Tower Model for Personalized Recommendations/Search

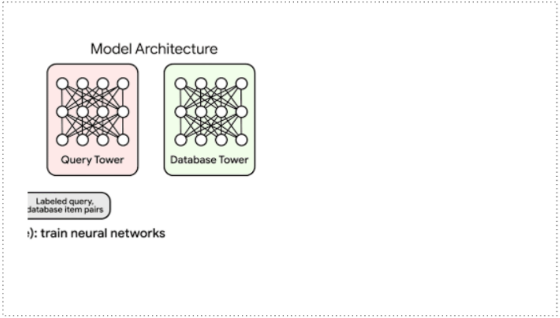


You should provide both positive and negative pairs, where the user and item interacted and where they didn't interact, respectively. Training produces 2 models: an item encoder model and a user encoder model.

[Image from Yu et al]

Build the ANN Index on Items. Similarity Search with user queries on it.





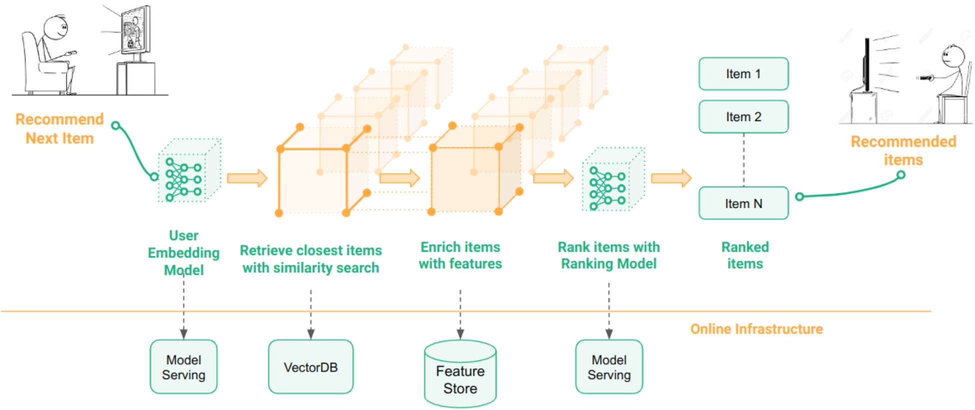
1. Encode and insert all the Items into your Approximate Nearest Neighbor (ANN) Index in your VectorDB
2. Now you can search for the nearest Item to the user query by encoding the user query and searching for the most similar items in the ANN Index

Source: https://cloud.google.com/blog/products/ai-machine-learning/vertex-matching-engine-blazing-fast-and-massively-scalable-nearest-neighbor-search

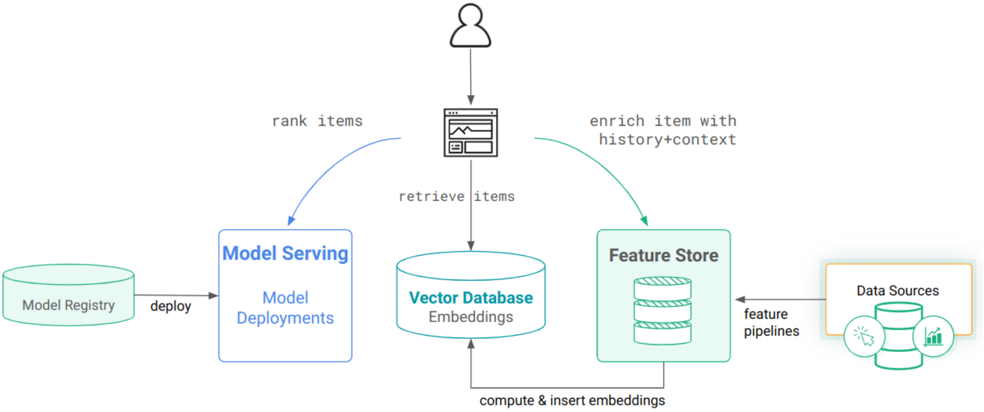
Build the ANN Index on Items. Similarity Search with user queries on it.

Click here to show animated insertions and lookups in a ANN with a Two-Tower Model

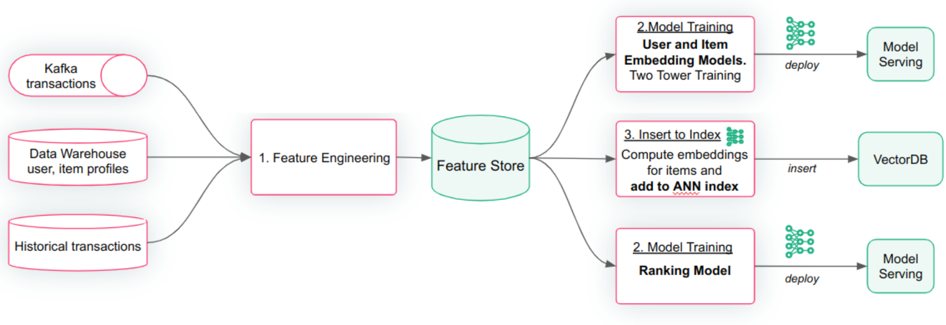
Real-Time Retrieval and Ranking



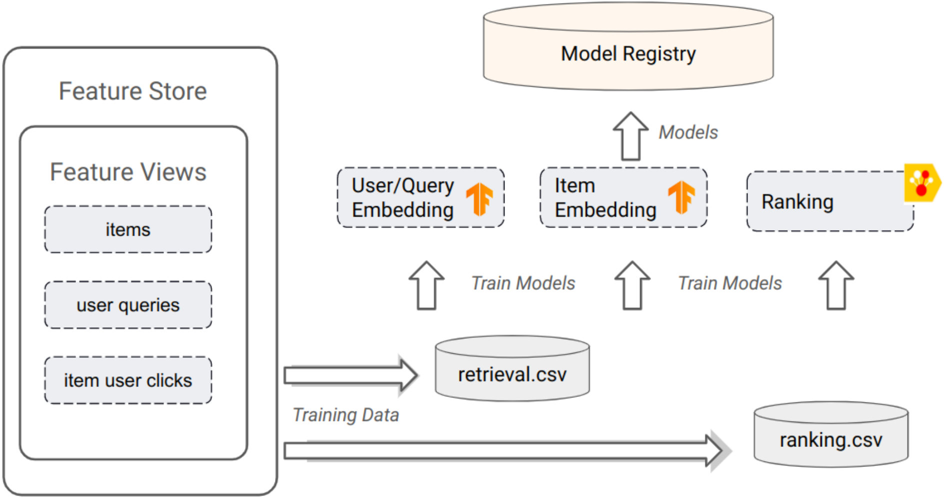
Real-Time Retrieval and Ranking Online Infrastructure



Offline Infrastructure for Retrieval and Ranking



Model Training for Embeddings and Ranking Model



Extended Retrieval and Ranking Architecture

Embeddings, Retrieval, Filtering, Ranking

User/Query & Item Embeddings → **Retrieval** → **Filtering** → **Ranking**

User/Query & Item Embeddings

Jointly train with two-tower model: User/query embedding Item embedding models

Built Approx Nearest Neighbor (ANN) Index with items and item embedding model.

Retrieval

Retrieve candidate items based on the user embedding from the ANN Index – similarity search

Filtering

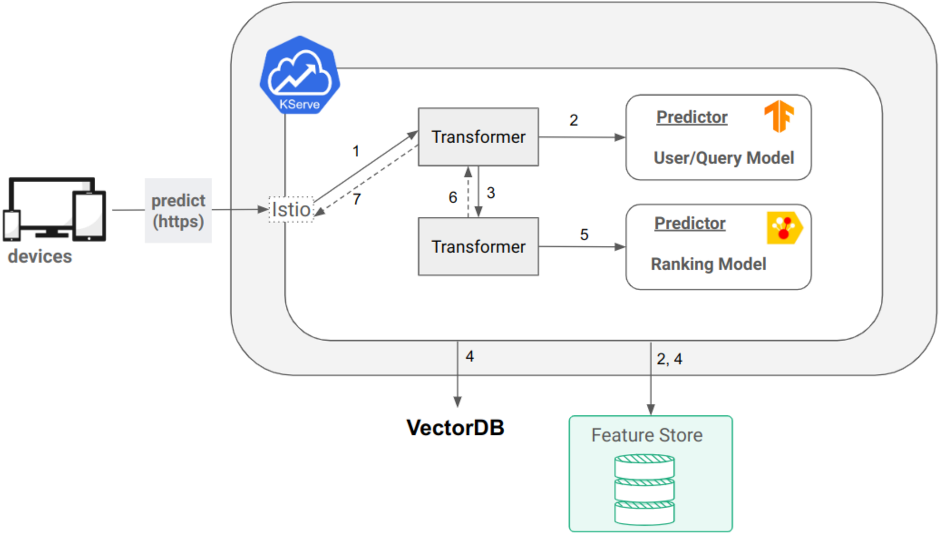
Remove candidate items for various reasons:

- underage user
- item sold out
- item bought before
- item not available in user's region

Ranking

With a ranking model, score all the candidate items with both user and item features, ensuring, candidate diversity.

Model Serving with VectorDB and Feature Store



References

- Real-time machine learning: challenges and solutions by Chip Huyen
- Concept Drift by FastForwardLabs
- Scale faster with less code using Two Tower with Merlin by Nvidia