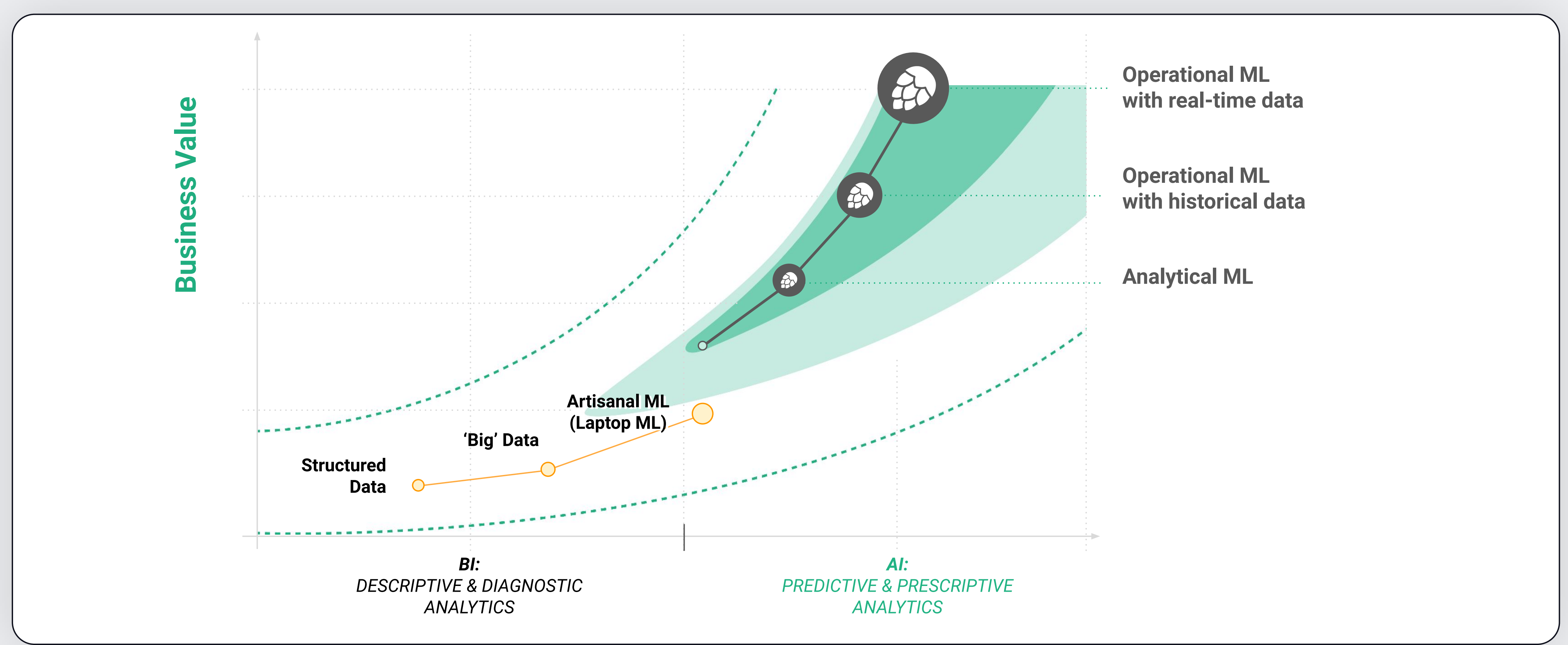
Real-Time Search and Recommendation at Scale using Embeddings and Hopsworks

Jim Dowling
CEO, Hopsworks

ORGANIZED BY databricks

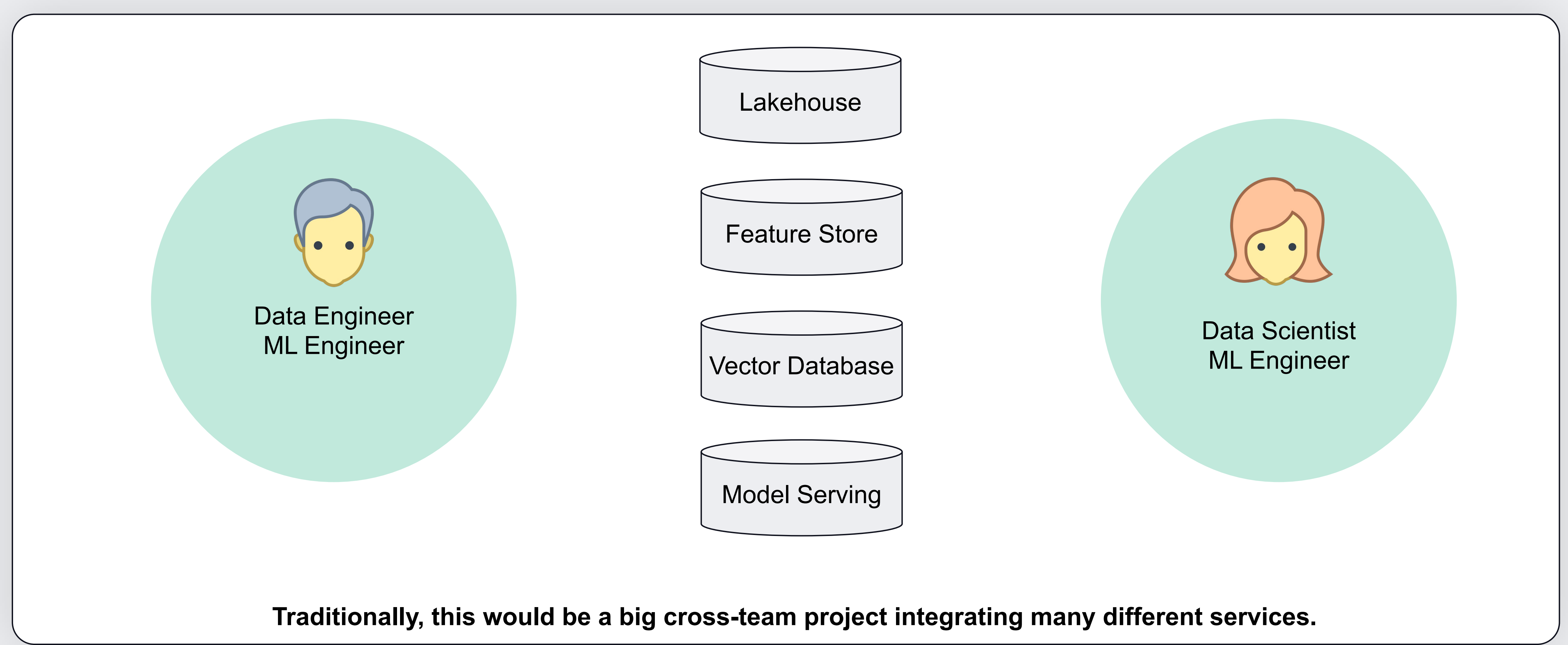
The Road to Value with Data and AI

Personalized Search/Recommendations is at the Highest level



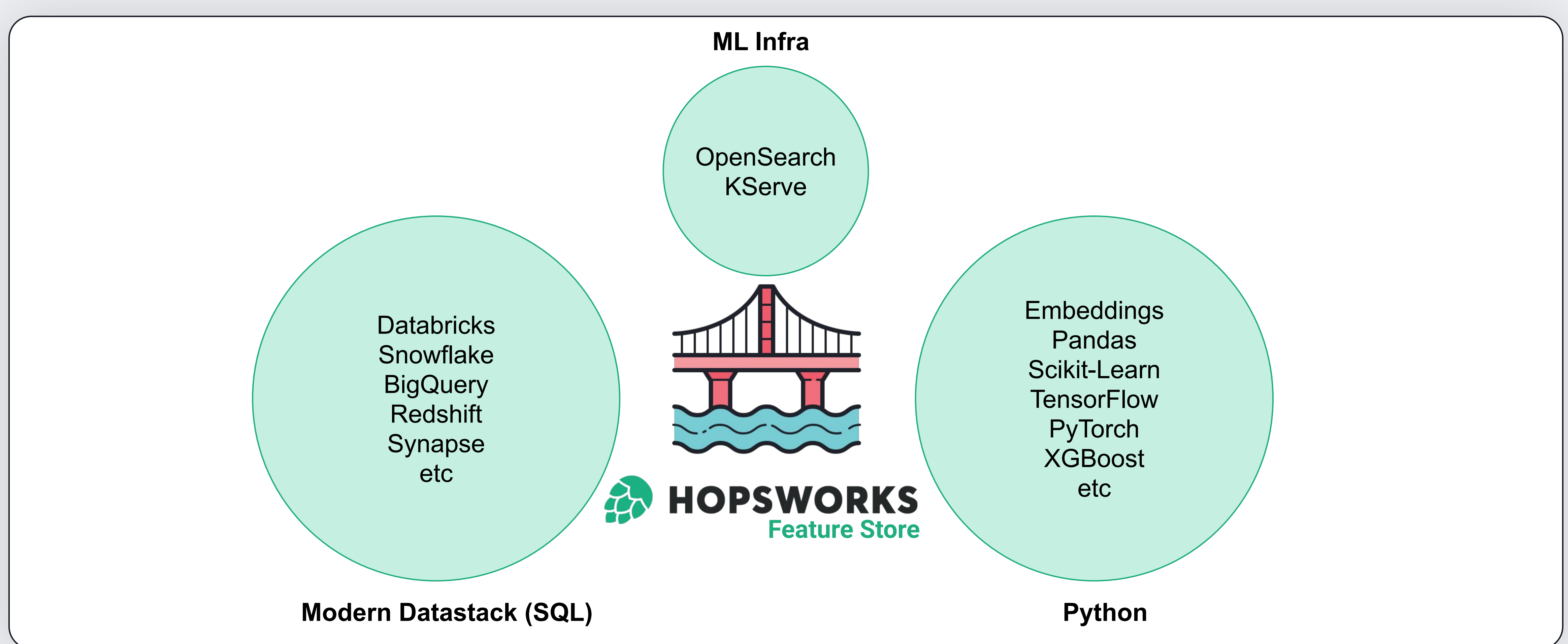
How can a Data Scientist build it in Python?

Python Only for Retrieval and Ranking



Hopsworks

Python-Centric: Feature Store, KServe, Vector DB all-in-one

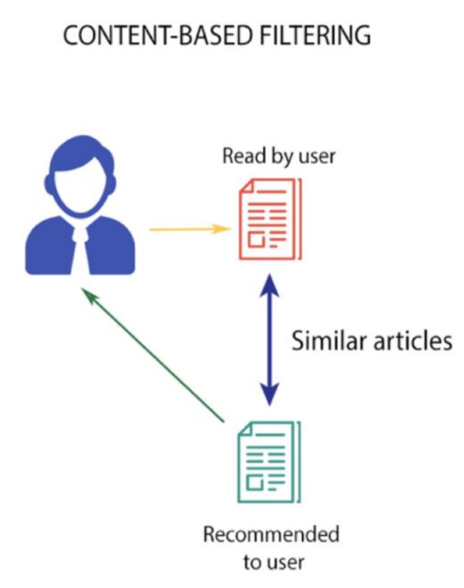


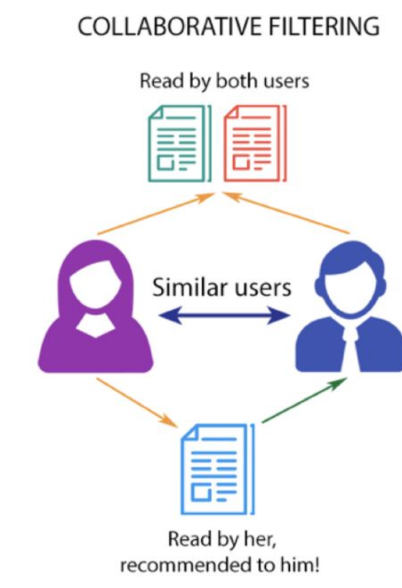
Classes of Recommender System

Where do the recommendations come from?

Multi-purpose

1. Collaboration-based recommendations are based on user behavior.
2. Content-based recommendations are based on item metadata.
3. Item-to-item (i2i) recommendations
   - given an item, recommend similar items
4. In user-to-item (u2i), given a user, we recommend items
5. i2i and u2i recommendations are dominant for user-centric websites





Well known online recommender services

Batch Recommender Systems

"It's digital crack", Andrej Karpathy

Facebook posts
Batch
Collaborative filtering
Content-based

Spotify Weekly
Batch
Collaborative filtering
Content-based

Netflix movies
Batch
Collaborative filtering
Content-based

MovieLens
Batch
Collaborative filtering
Content-based

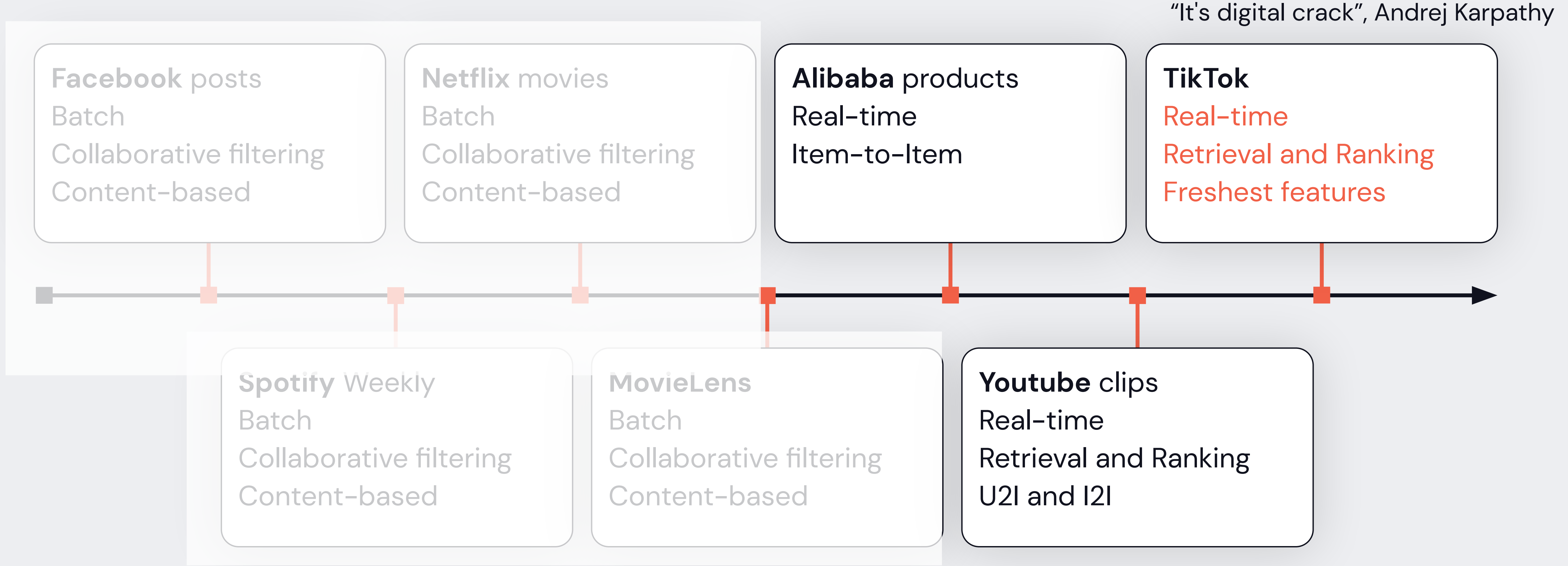
Alibaba products
Real-time
Item-to-Item

Youtube clips
Real-time
Retrieval and Ranking
U2I and I2I

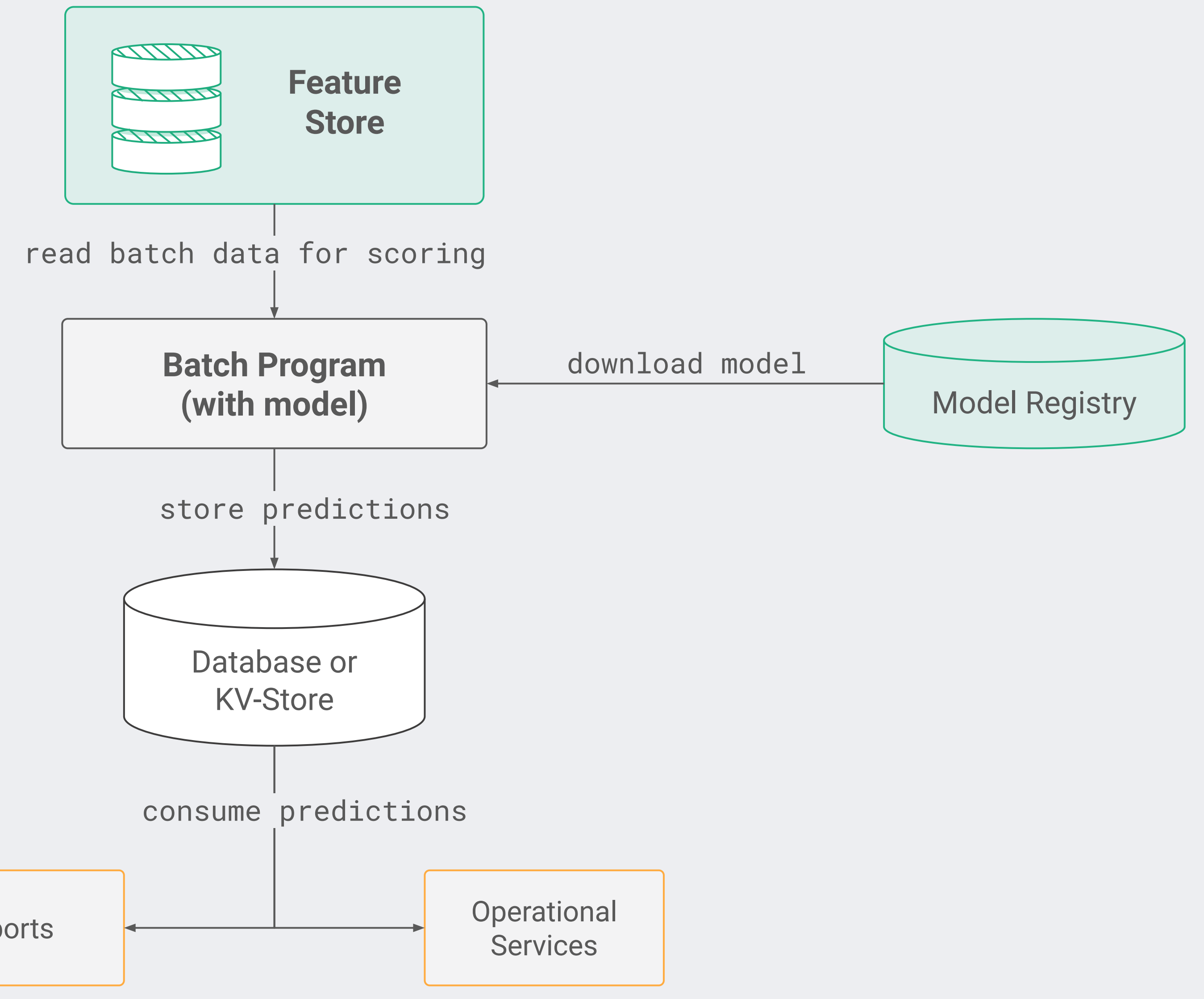
TikTok
Real-time
Retrieval and Ranking
Freshest features

Well known online recommender services

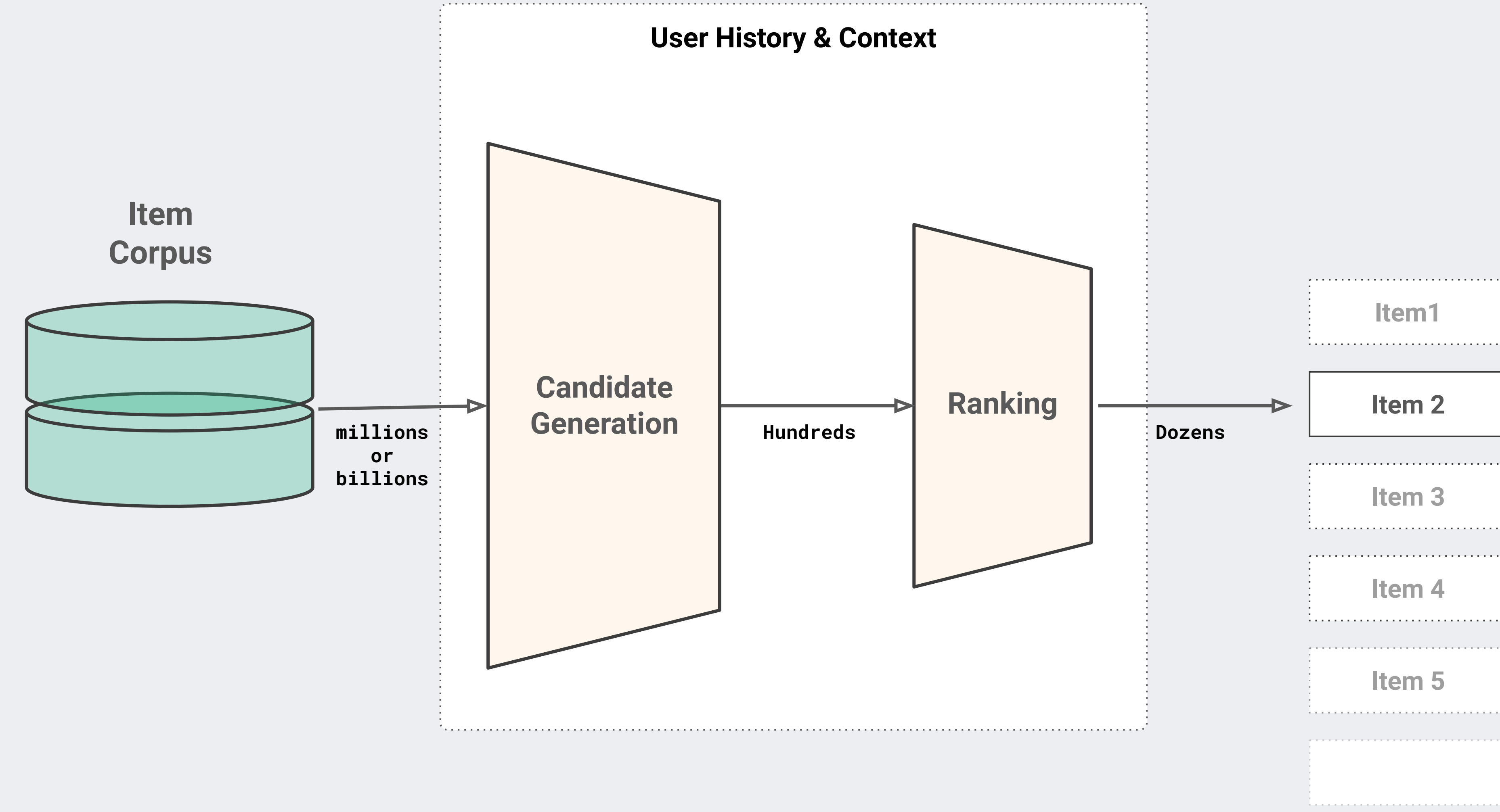
Real-time Recommenders



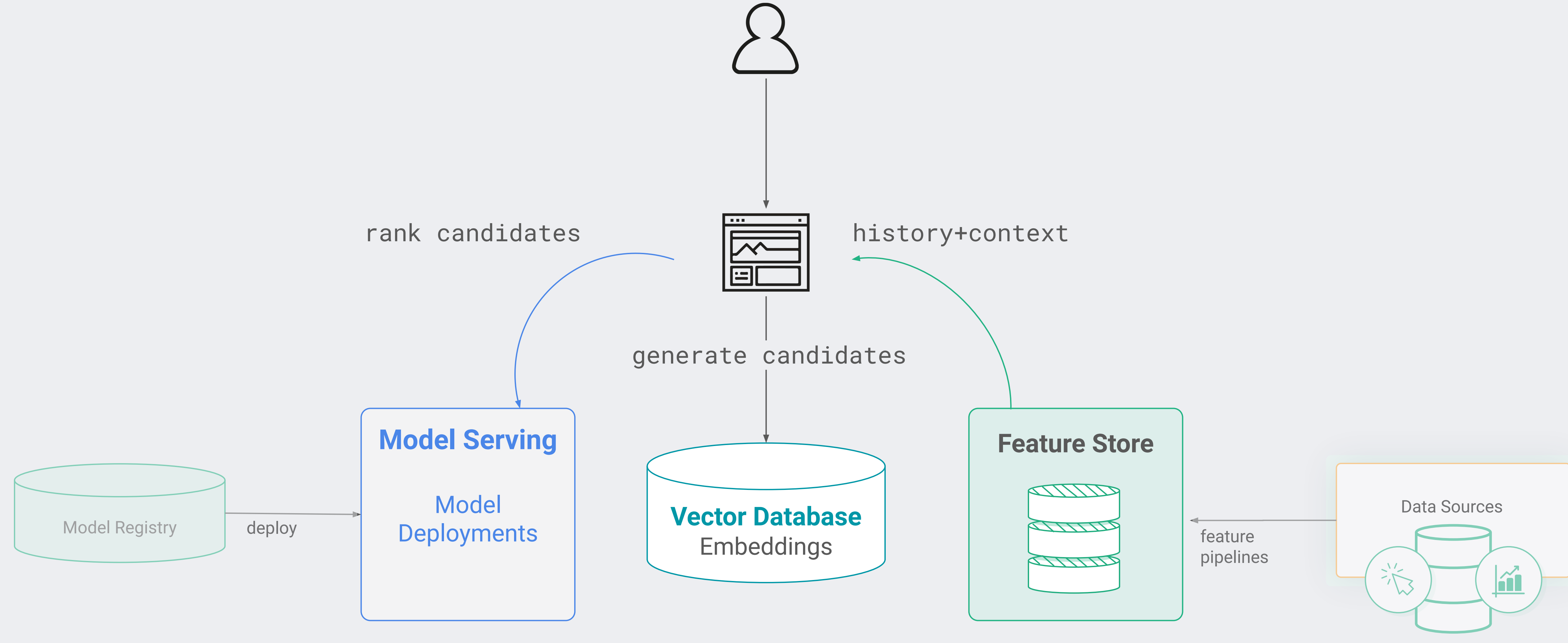
Batch Recommender Service



Real-time Recommender Service



Real-time Recommender Service - Retrieval and Ranking



Retrieval/Ranking Arch for Recommendations

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:

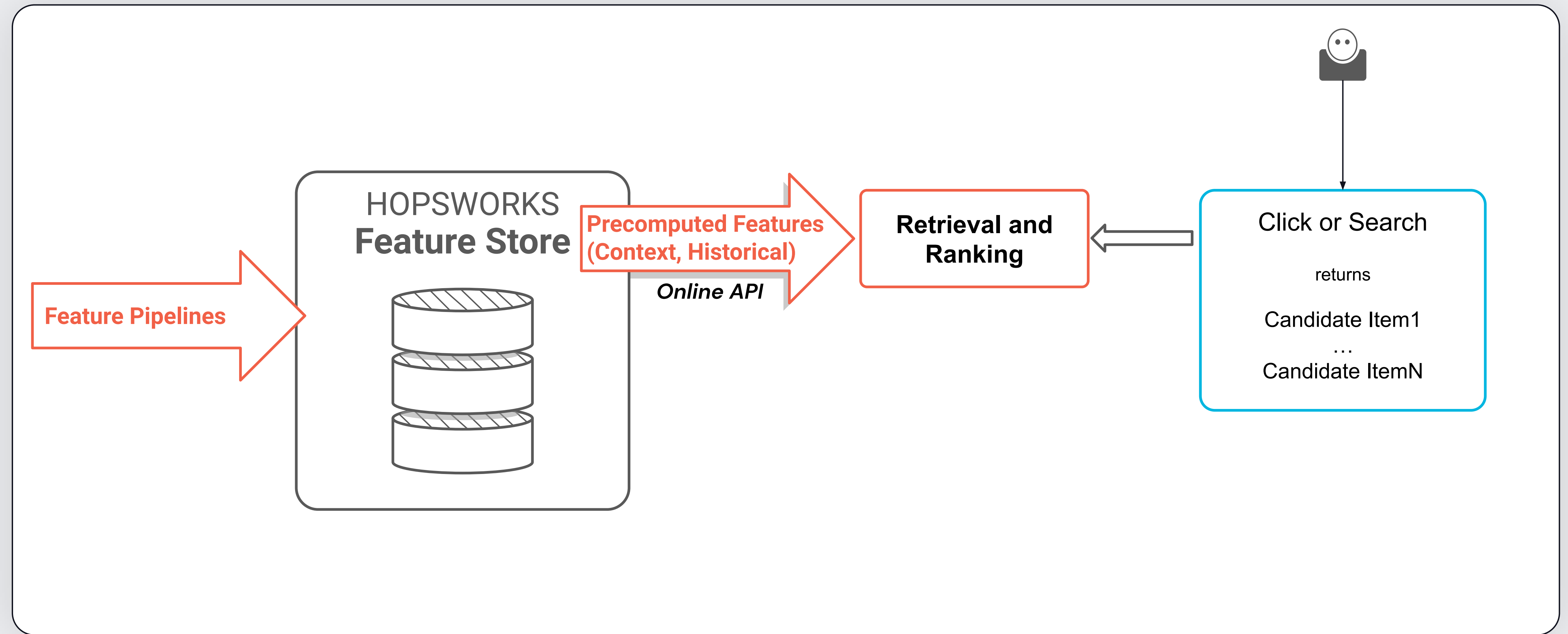
- underage user
- item sold out
- item bought before
- item not available in user’s region

Ranking

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

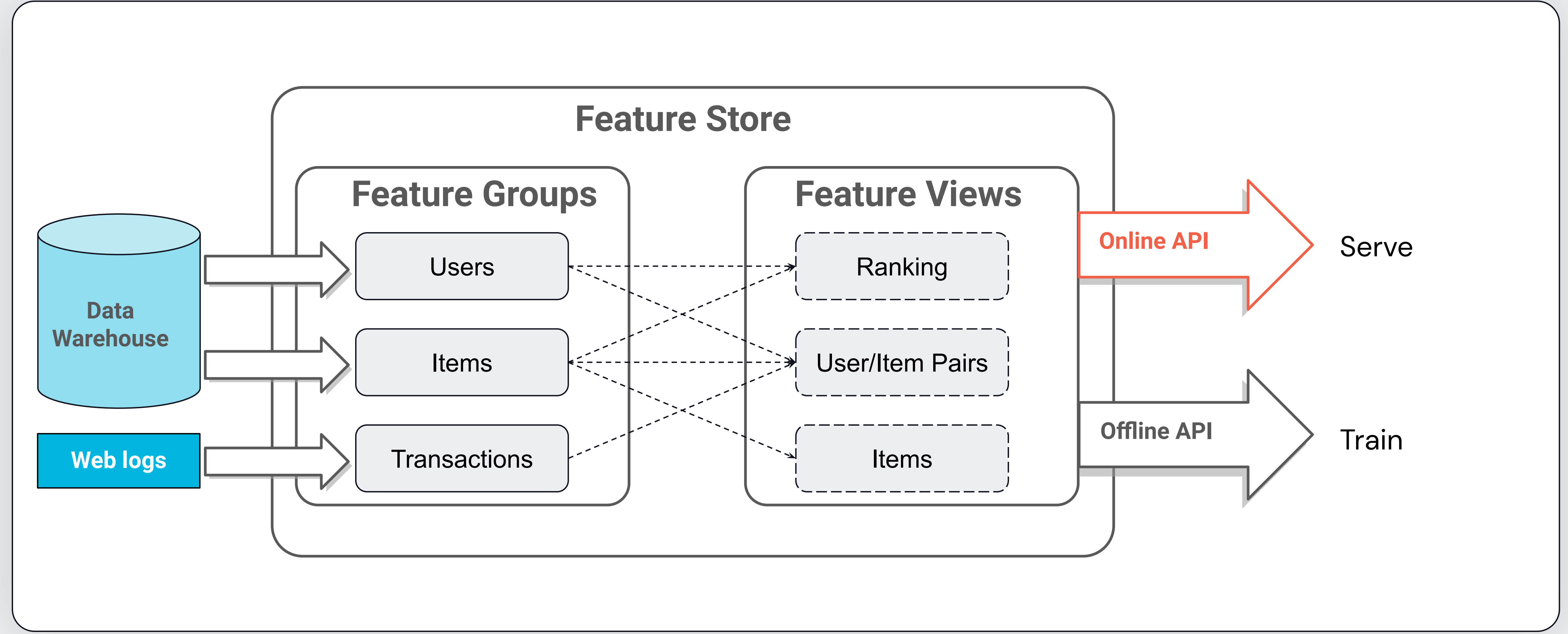
Feature Store and Retrieval/Ranking

Context and History for Real-time Models



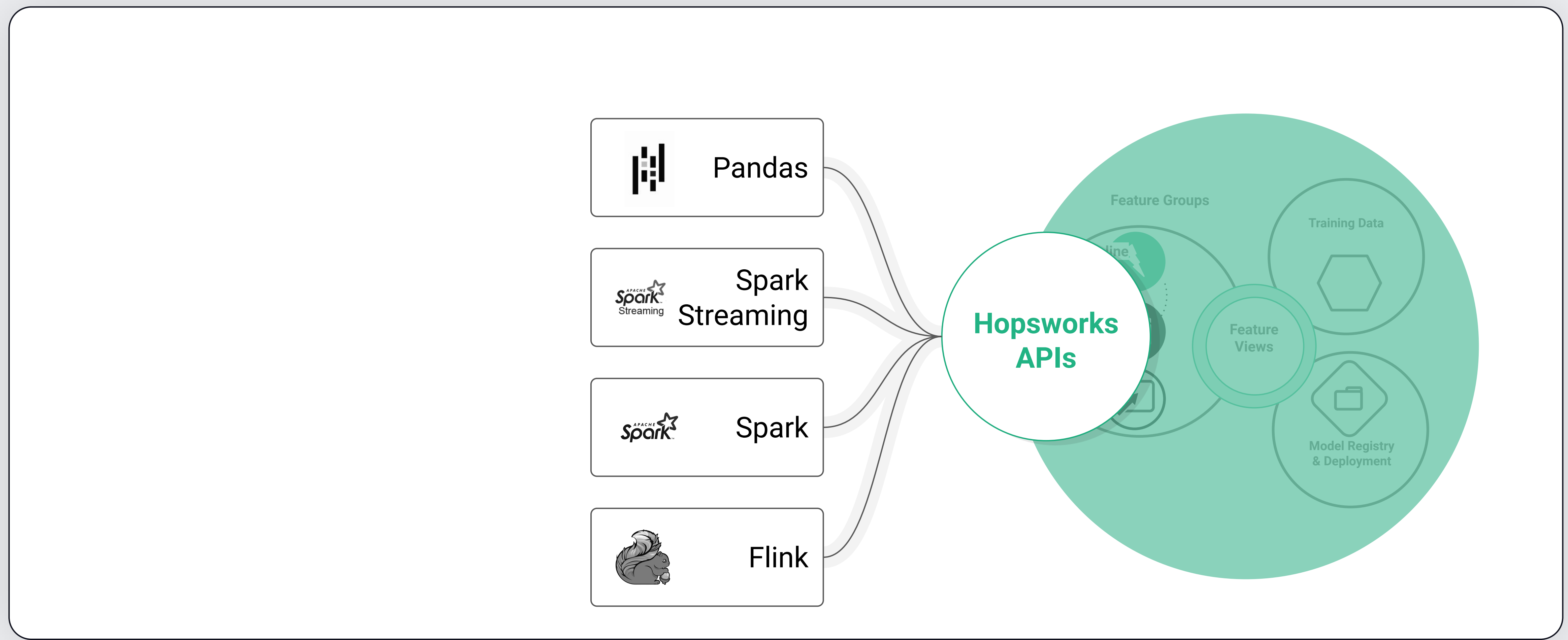
Inside the Feature Store

Write to Feature Groups, Read from Feature Views



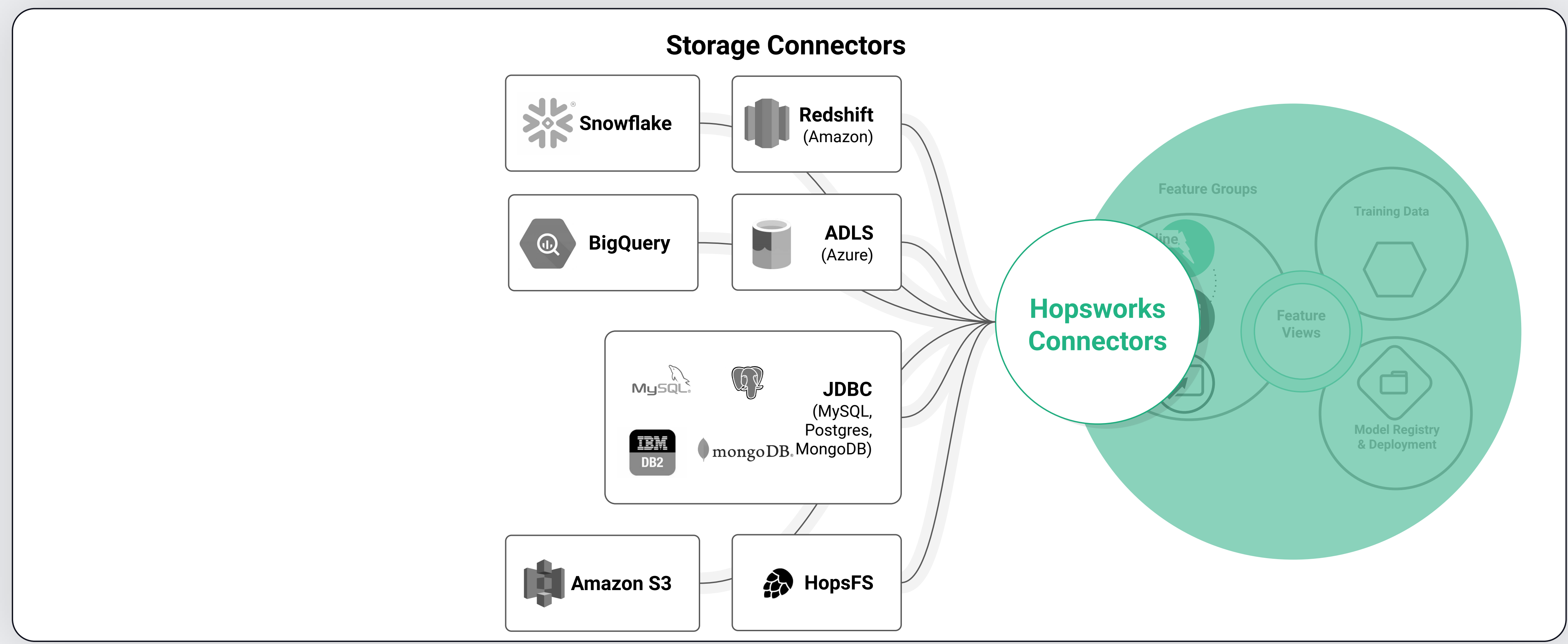
Writing to Feature Groups

APIs for writing data into Feature Groups



External Feature Groups

Mount tables from external data sources into Feature Store



Feature/Prediction Logging

Needed to create training data

Log user-item interactions as training data for our two-tower model and ranking model.

Retail Website

Search

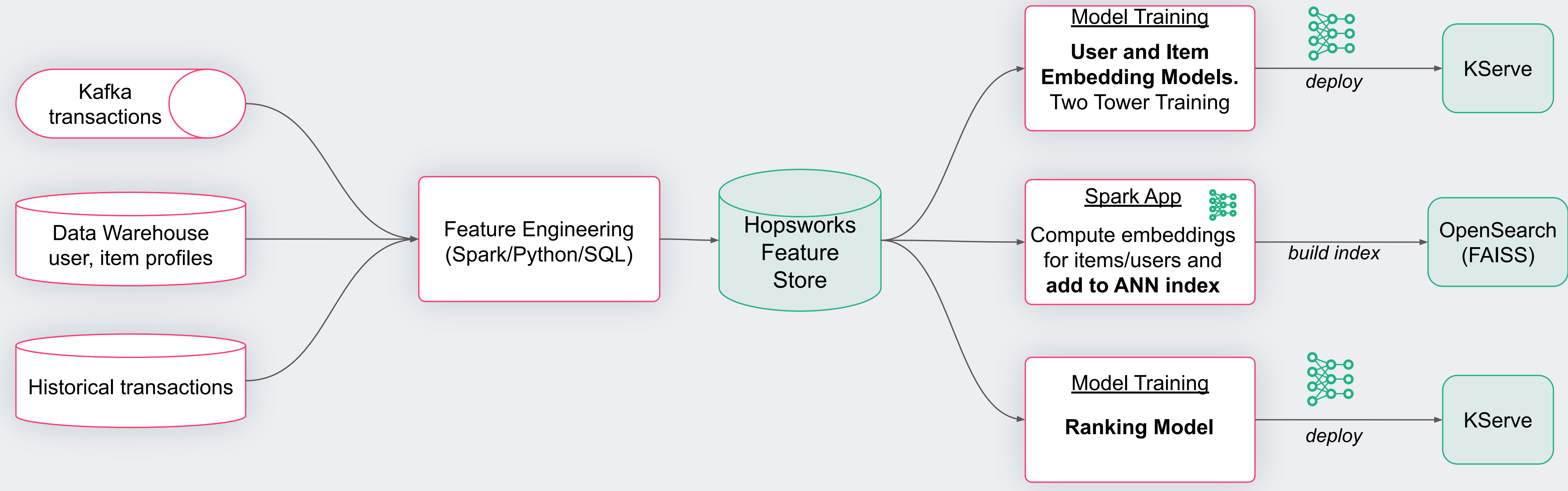
Item 1
Item 2
Item 3
Item 4

Click 2
Click 3
Purchase 3

Item	Features	Score
Item 1	Features	Score: 0
Item 2	Features	Score: 1
Item 3	Features	Score: 5
Item 4	Features	Score: 0

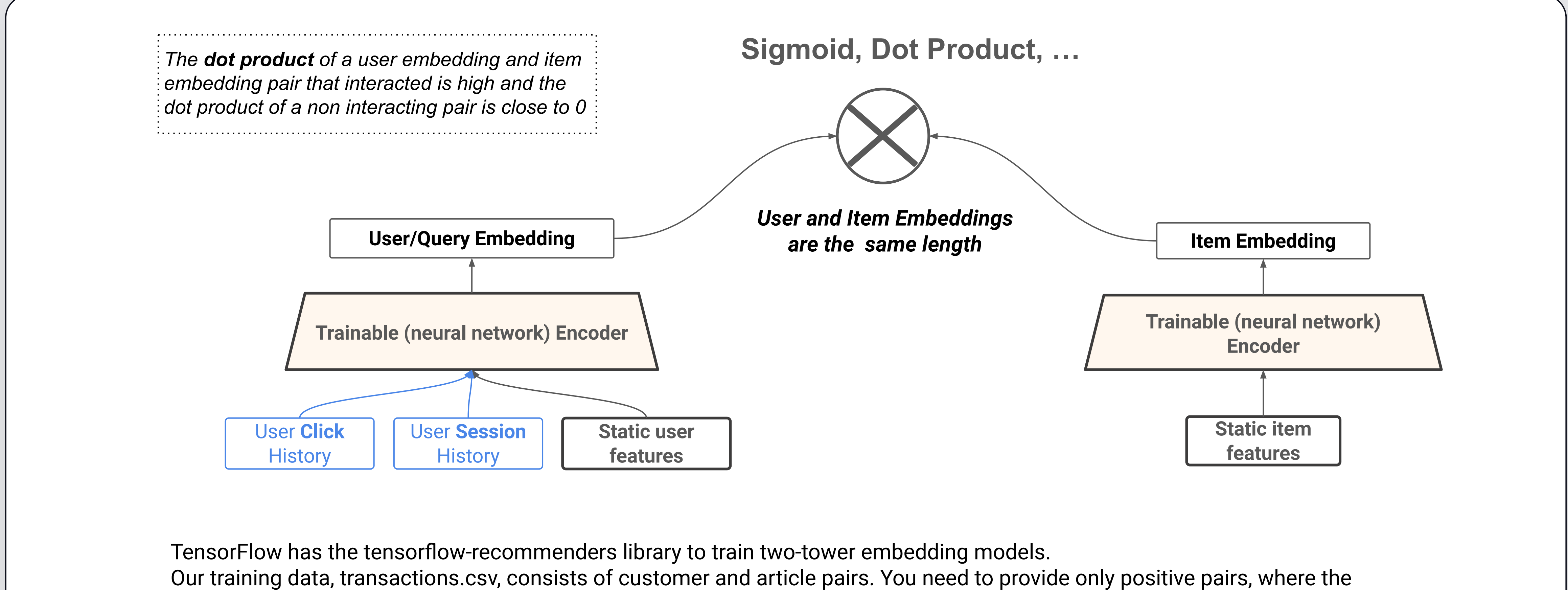
Offline Infrastructure

Collect user, item and transaction (clicks/searches) data



Network Architecture for Two-Tower Model

User/query tower and item tower



TensorFlow has the tensorflow-recommenders library to train two-tower embedding models.
Our training data, transactions.csv, consists of customer and article pairs. You need to provide only positive pairs, where the customer purchased an article. Training produces 2 models: an item encoder model and a user encoder model.

[Image from Yu et al]

Ranking Model

Model should predict best order with user/item features

Input: a set of instances

$$X = \{x_1, x_2, \ldots, x_n\}$$

Output: a rank list of these instances

$$\hat{Y} = \{x_{r_1}, x_{r_2}, \ldots, x_{r_n}\}$$

Ground truth: a correct ranking of these instances

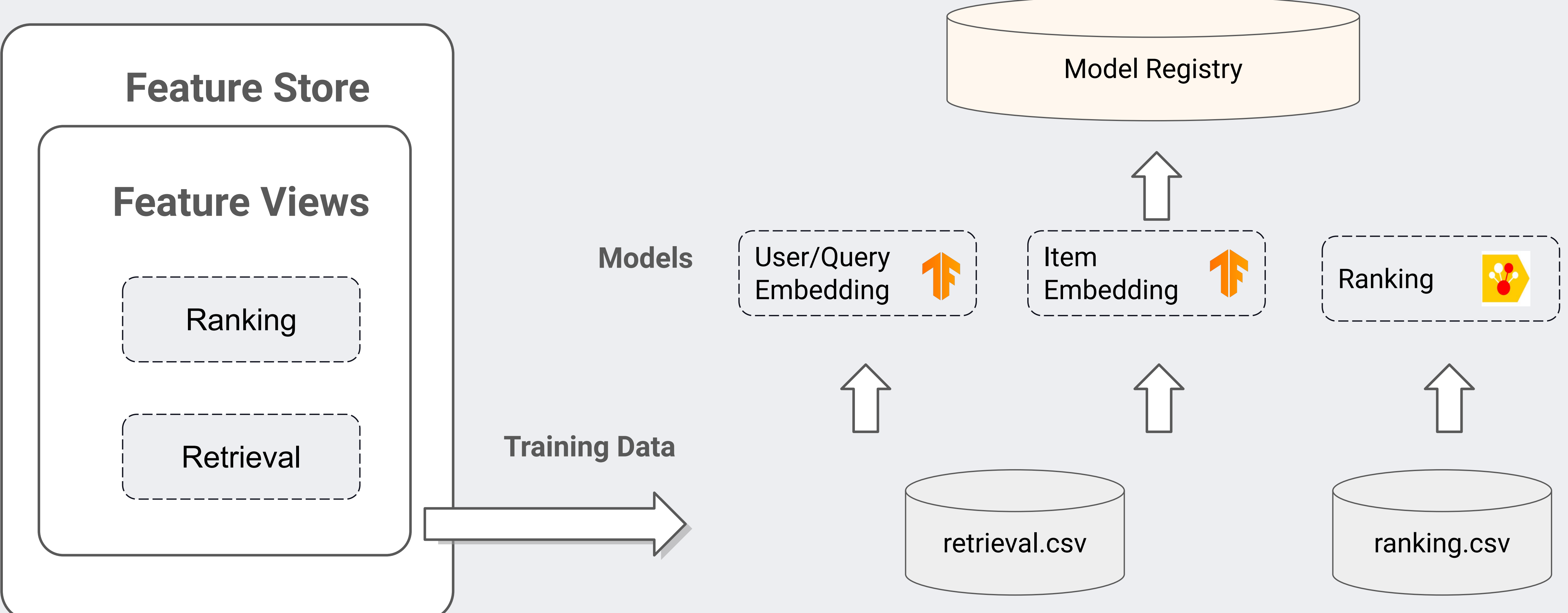
$$Y = \{x_{y_1}, x_{y_2}, \ldots, x_{y_n}\}$$

Each instance (user-item pair) is represented with a list of features, retrieved from the feature store.

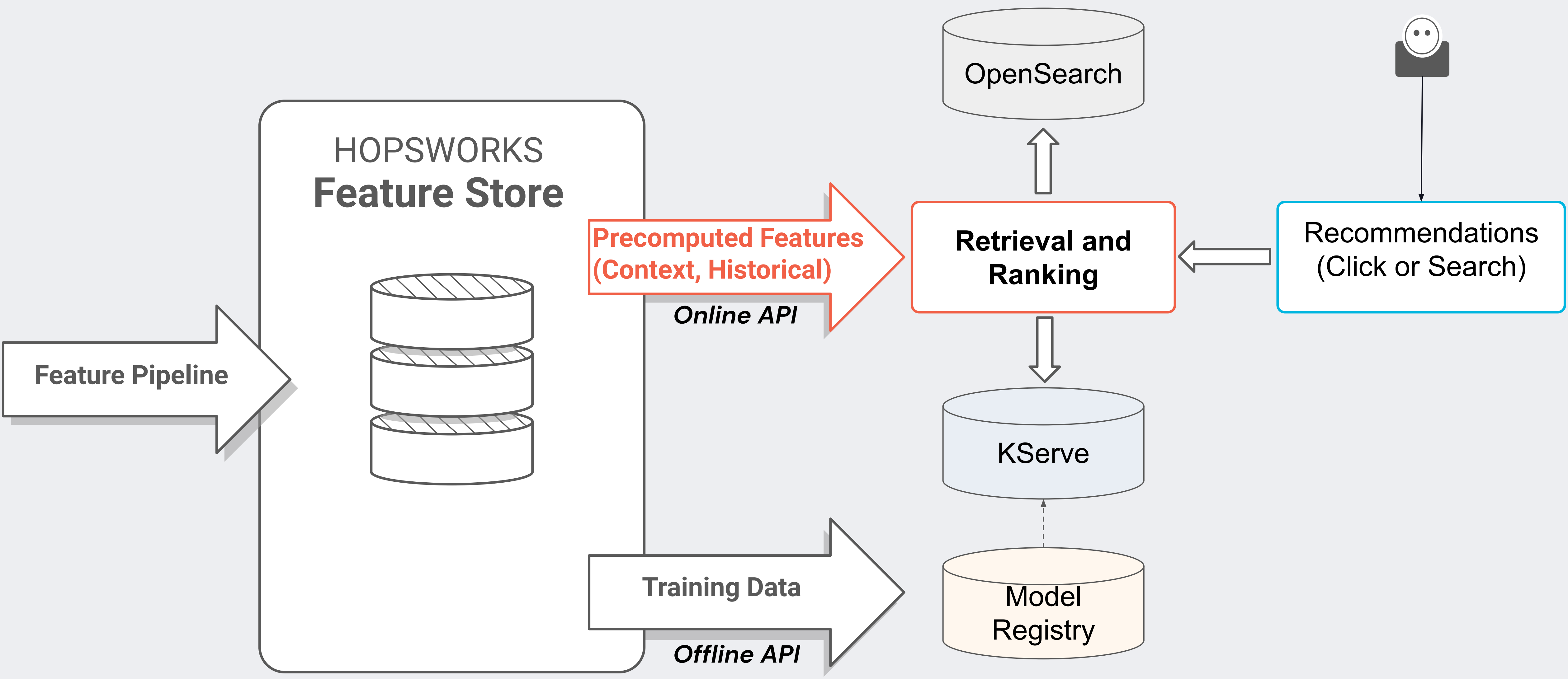
Training data is the user-item features and the label is the relevance ratings.

Ranking models should be fast - low latency to rank 100s of candidates, so decision trees are popular.

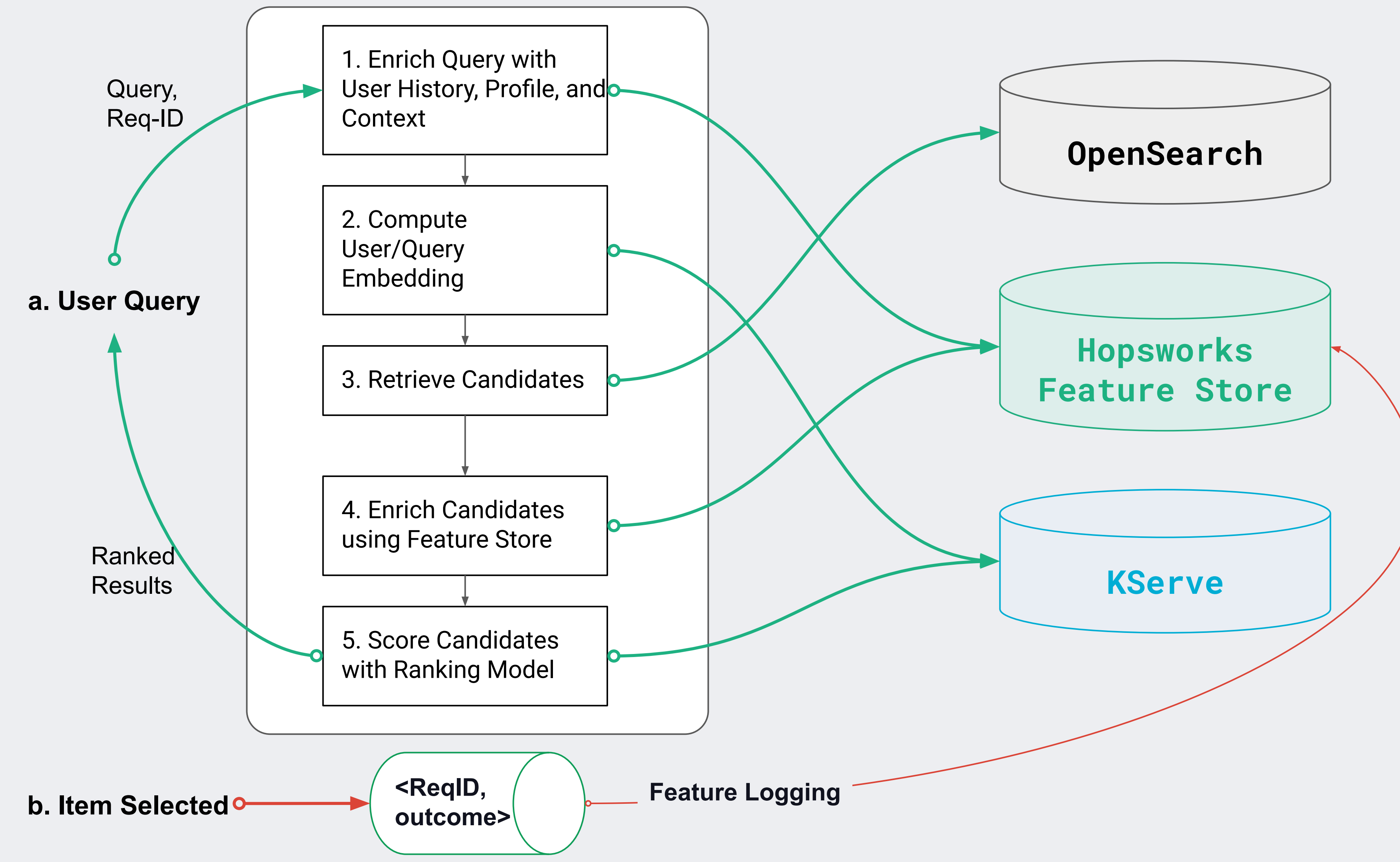
Training Models



Hopsworks Retrieval and Ranking



Hopsworks Ranking and Retrieval



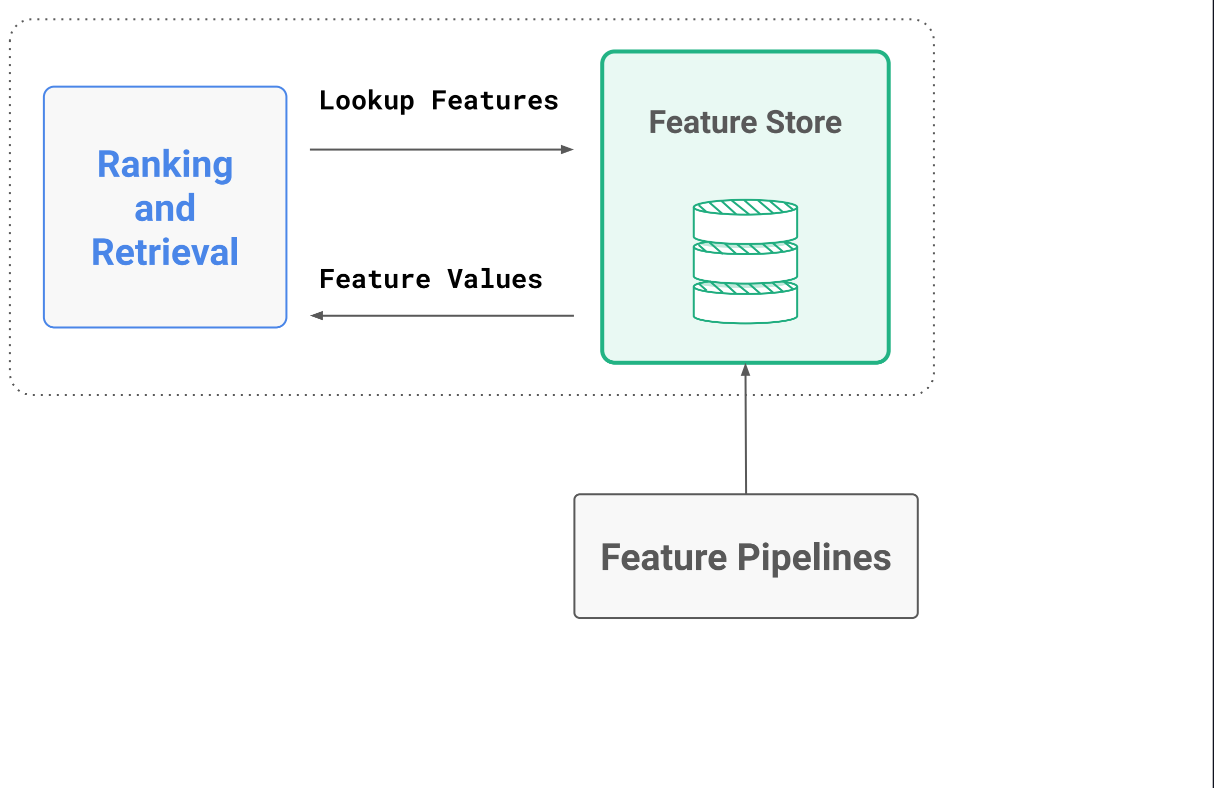
DATA+AI
SUMMIT 2022

Benchmarking

Retrieval and Ranking at Scale with Spotify

Goal:
Support Spotify Personalized Search in a Retrieval and Ranking Architecture.
Benchmark the highest throughput, lowest latency key-value stores to identify one that could scale to handle millions of concurrent lookups per second on Spotify's workloads.

Systems:
Aerospike and **RonDB** were identified as the only systems capable of meeting the triple goals of High Throughput, Low Latency, and High Availability. Other databases such as Redis, Cassandra, BigTable were not considered for availability or latency or throughput reasons.



Benchmarking

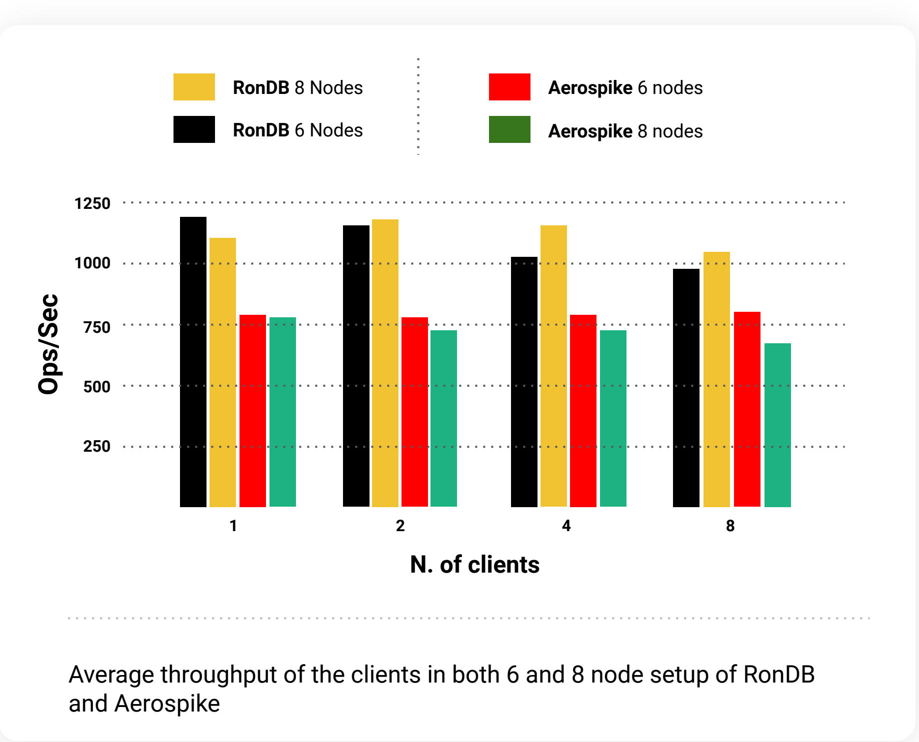
Experiment setup

Node Type	GCP Instance Type	Virtual CPUs	Memory	Disk Size	Disk Type
MySQL Servers	n1-standard-2	2	7.5GB	120GB	pd-ssd
NDB Management Node	e2-standard-16	16	64GB	120GB	pd-ssd
NDB Data Nodes	n1-highmem-32	32	208GB	408GB	pd-ssd
Aerospike Nodes	n1-highmem-32	32	208GB	408GB	pd-ssd
Java Client Nodes	e2-standard-16	16	64GB	120GB	pd-ssd

Hardware Benchmark Setup on GCP: RonDB (NDB) vs Aerospike. The Java Client nodes are the clients performing the reads/writes on the Data Nodes. When the cluster is provisioned with 8 RonDB (NDB) data nodes, it has 832GB of usable in-memory storage, when a replication factor of 2 is used.

Benchmarking

Throughput

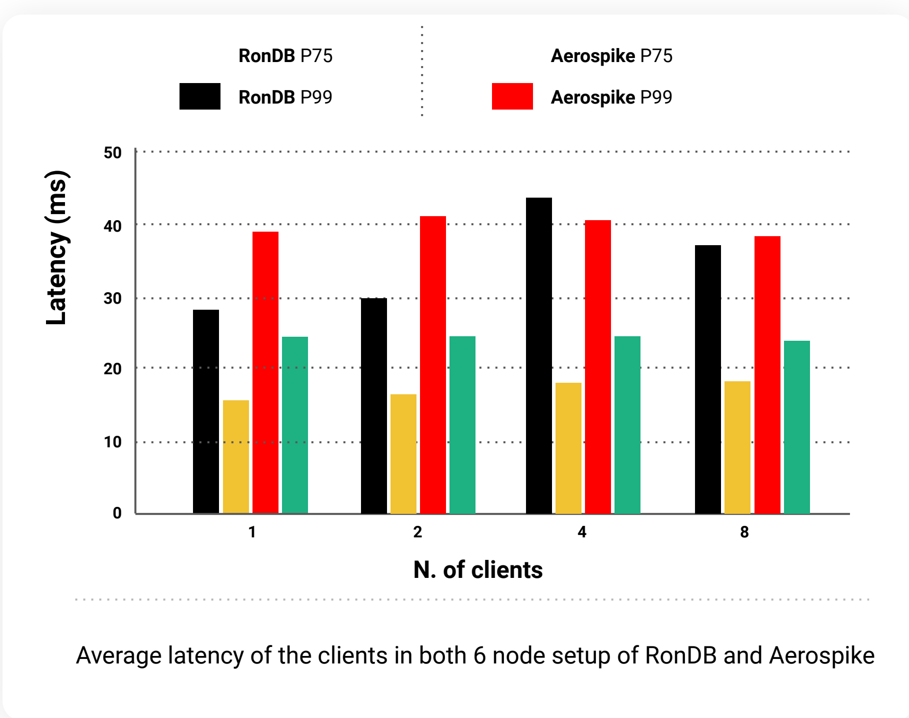


Average throughput of the clients in both 6 and 8 node setup of RonDB and Aerospike

Throughput: higher is better. Each feature store operation was a batch of 250 key-value lookups, meaning with 8 clients for a 8-node RonDB cluster, there are >2m lookups/sec.

Benchmarking

Latency



Average latency of the clients in both 6 node setup of RonDB and Aerospike

Latency: lower is better. Each feature store operation was a batch of 250 key-value lookups. So, for RonDB, the P99 when performing 250 primary operations in a single transaction is under 30ms.

Benchmarking

Spotify Online Feature Store Comparison

RonDB **35% Higher Throughput**
RonDB **30% Better Latency**

Based on Public Report from Spotify
comparing Aerospike and RonDB (NDB Cluster) as Feature Stores

http://kth.diva-portal.org/smash/get/diva2:1556387/FULLTEXT01.pdf

DEMO

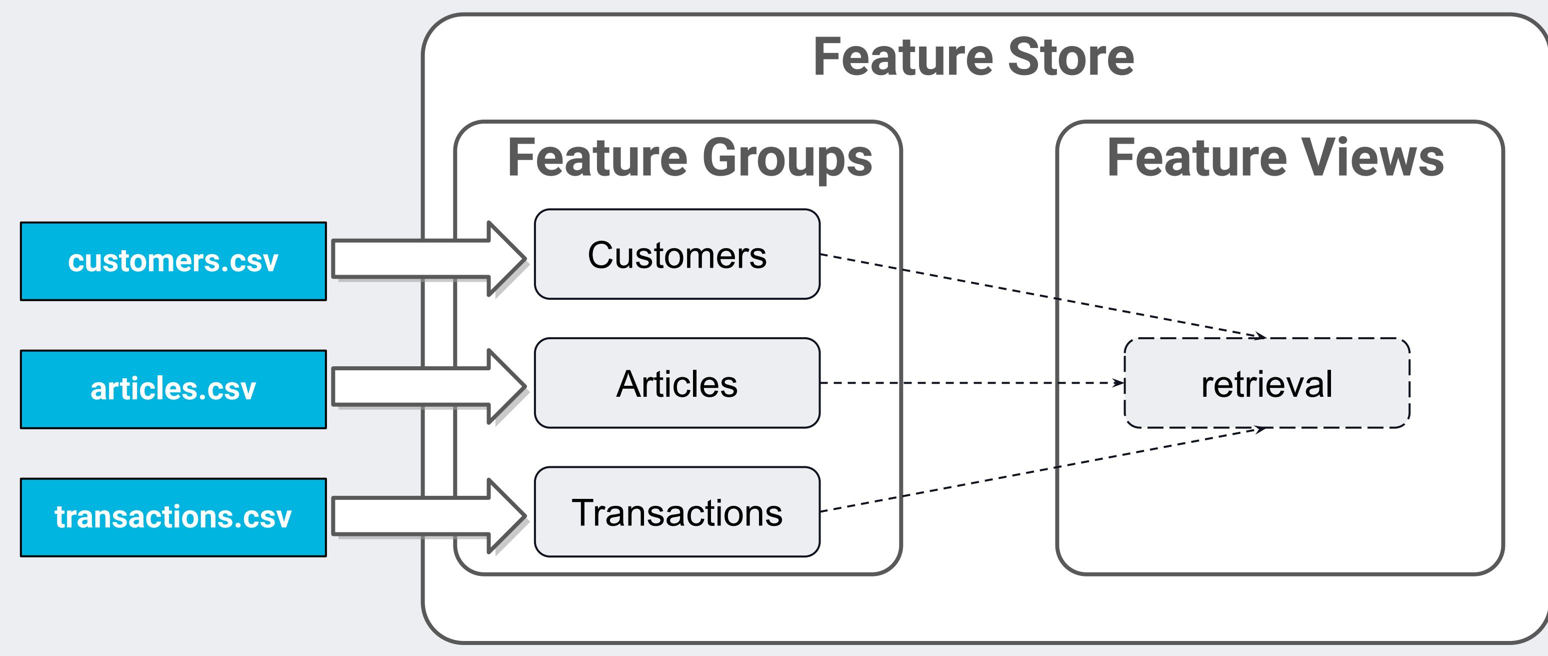
H&M Dataset from Kaggle

- articles.csv
- customers.csv
- transactions_train.csv

Notebook	Description
1_feature_engineering.ipynb	create feature groups for articles, customers, transactions
2a_create_retrieval_dataset.ipynb	create feature view for retrieval model (training data)
2b_train_retrieval_model.ipynb	train two-tower model - user and article embedding models
3_build_index.ipynb	build opensearch KNN index with embeddings for all articles
4a_create_ranking_dataset.ipynb	create feature view for retrieval model (training data)
4b_train_ranking_model.ipynb	train ranking model
5_create_deployment.ipynb	deploy models to KServe + glue code for Hopsworks, OpenSearch

https://github.com/logicalclocks/hopsworks-tutorials

H&M Dataset from Kaggle – Demo

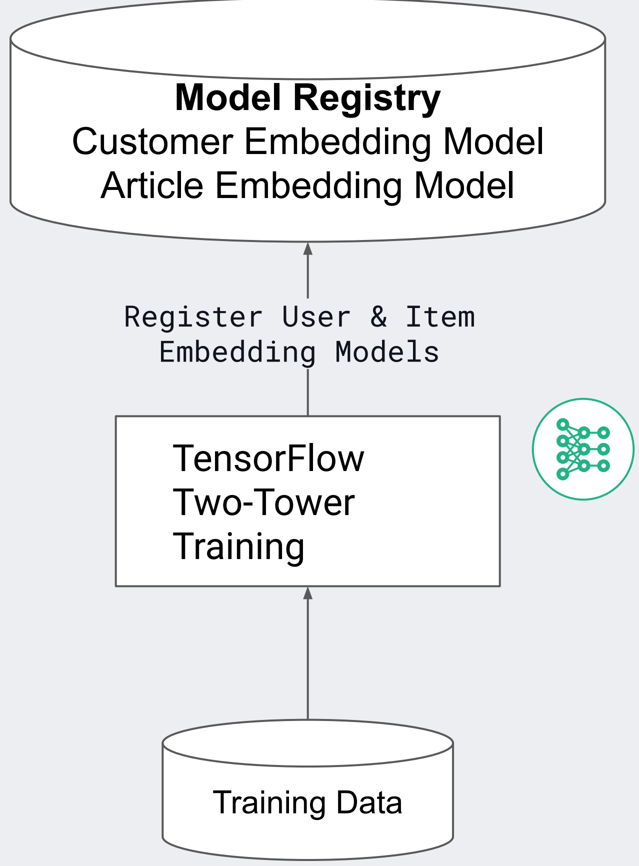


Transaction = (customer_id, article_id, timestamp, channel)

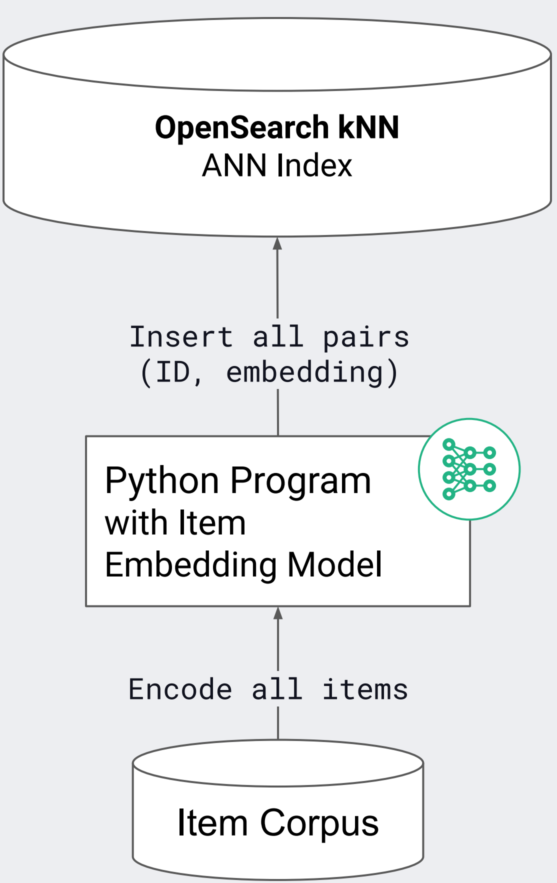
1_feature_engineering.ipynb

2a_create_retrieval_feature_views.ipynb

H&M Dataset from Kaggle – Demo

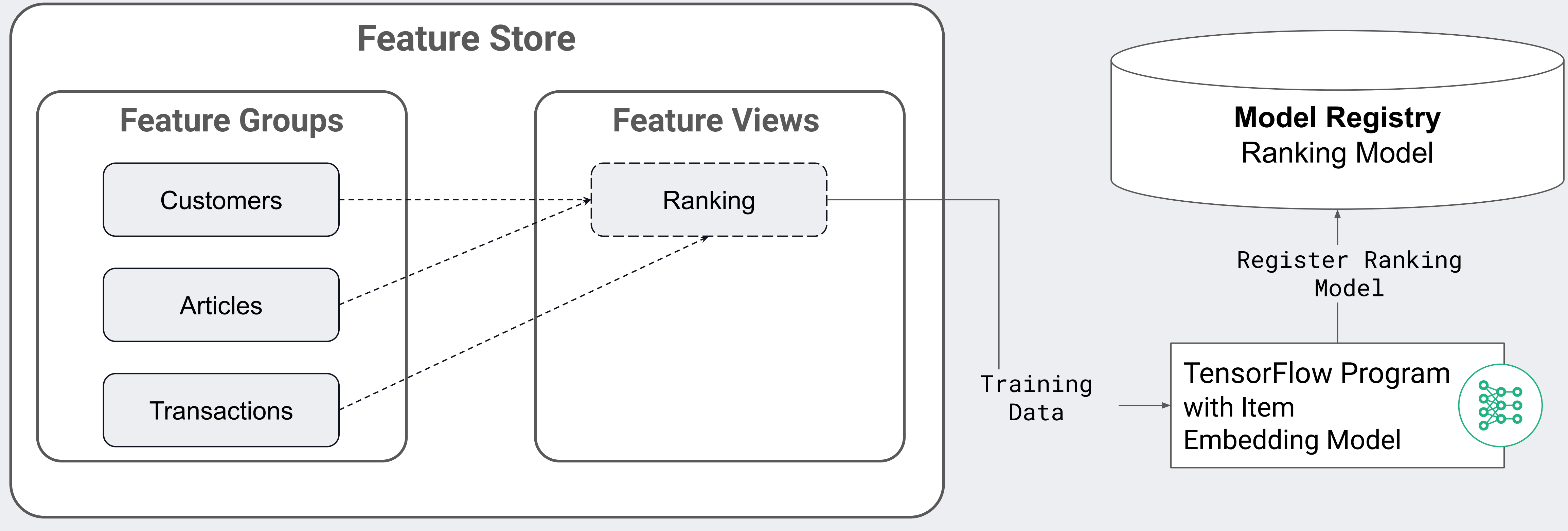


2b_train_retrieval_model.ipynb



3_build_index.ipynb

H&M Dataset from Kaggle – Demo

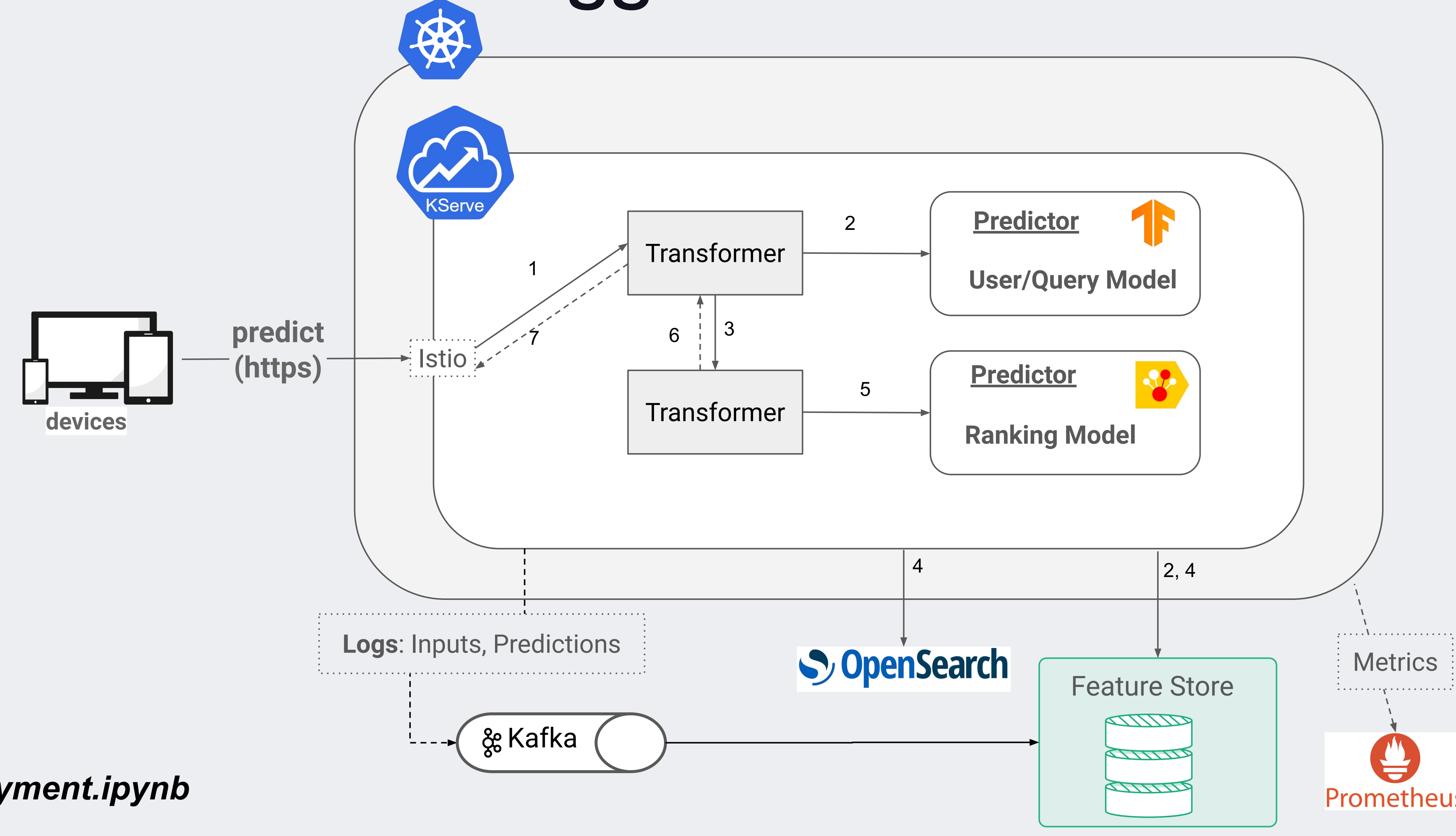


Transaction = (customer_id, article_id, timestamp, channel)

4a_create_ranking_feature_views.ipynb

4b_train_ranking_model.ipynb

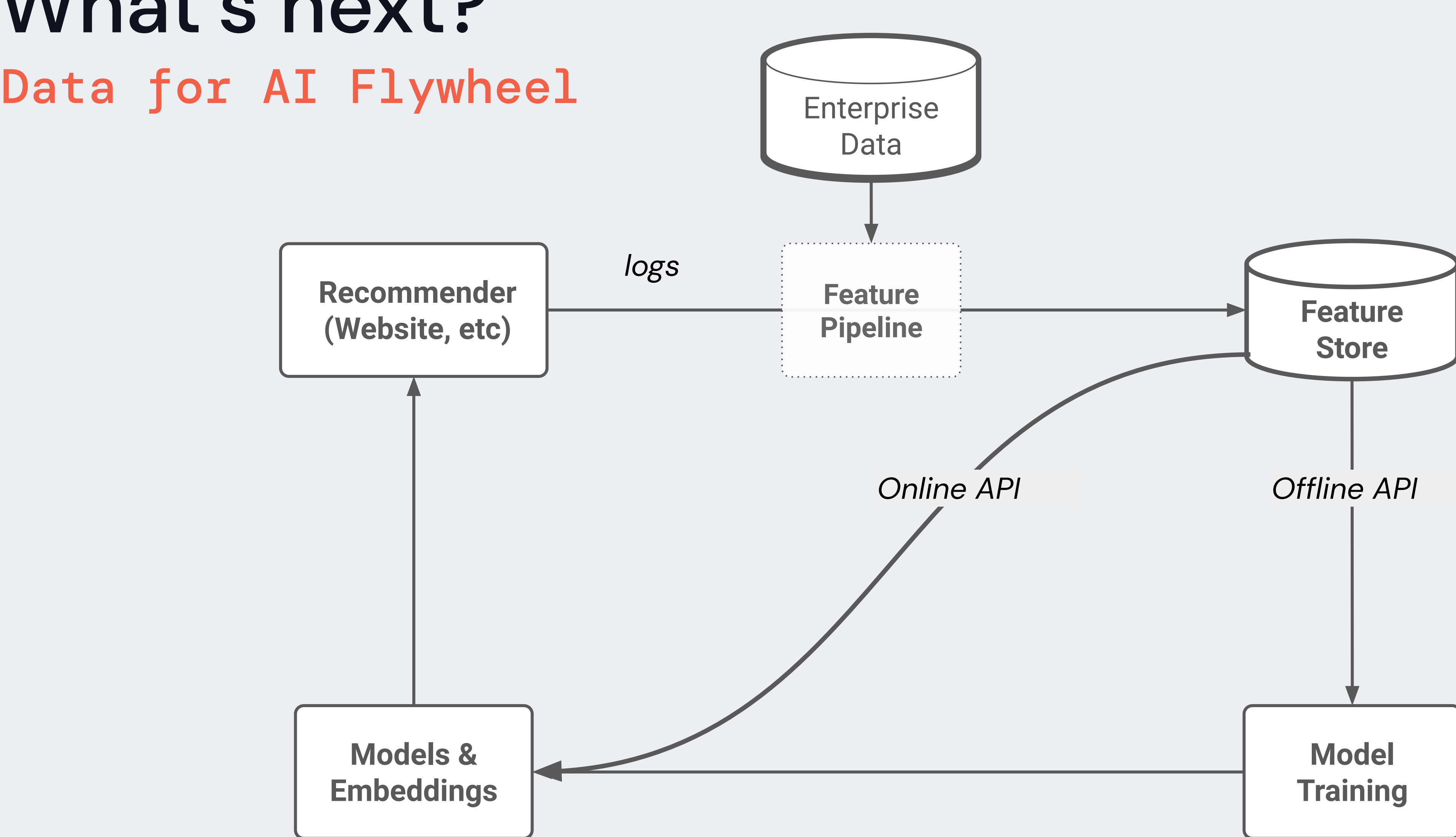
H&M Dataset from Kaggle – Demo



5_create_deployment.ipynb

What's next?

Data for AI Flywheel



Build Prediction Services, not just Models

Hopsworks Serverless

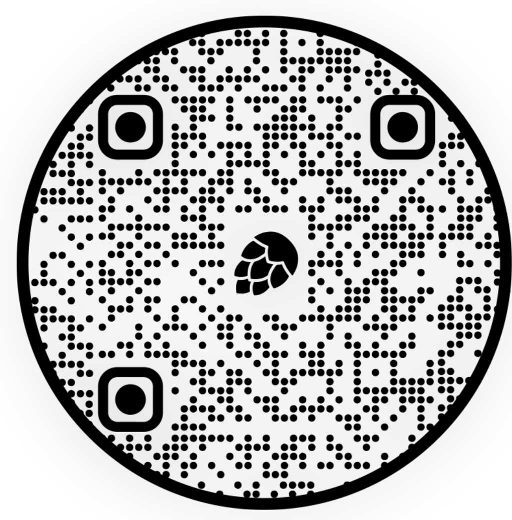
Now available at:

https://app.hopsworks.ai

Our Promise to you:

Free Forever

HOPSWORKS



www.hopsworks.ai

Efficiency
At Scale

Open &
modular

References

Hopsworks Feature Store

OpenSearch k-NN (Embeddings Store)

TFRanking tutorial

Augmented Two-Tower Embedding Model

Ranking and Filtering by Zhang