feathr

An open source, enterprise-grade, high-performance feature store

https://github.com/feathr-ai/feathr

- built at LinkedIn & Microsoft
- a Linux Foundation AI & Data Sandbox project
- Now natively integrated with Azure/AWS, and Databricks

Agenda

1. The Problem
2. The Solution
3. The Use Case
4. Summary

Survey in Forbes: Big data engineering for AI

Re: Building training sets + Cleaning and organizing data + Collecting datasets

82%

Time spent on data preparation

88%

Respondents said data preparation 'least enjoyable' part of data science

Like a music streaming app … for feature engineering

Music "Workflow"

Old Way

- Manually get music files from various sources
- Convert them to a format my device can play
- Load onto my device (different for home/car)
- Worry about storage, bitrate, compatibility

New Way

- **Just ask virtual assistant to play the song.**

ML Feature Workflow

Old Way

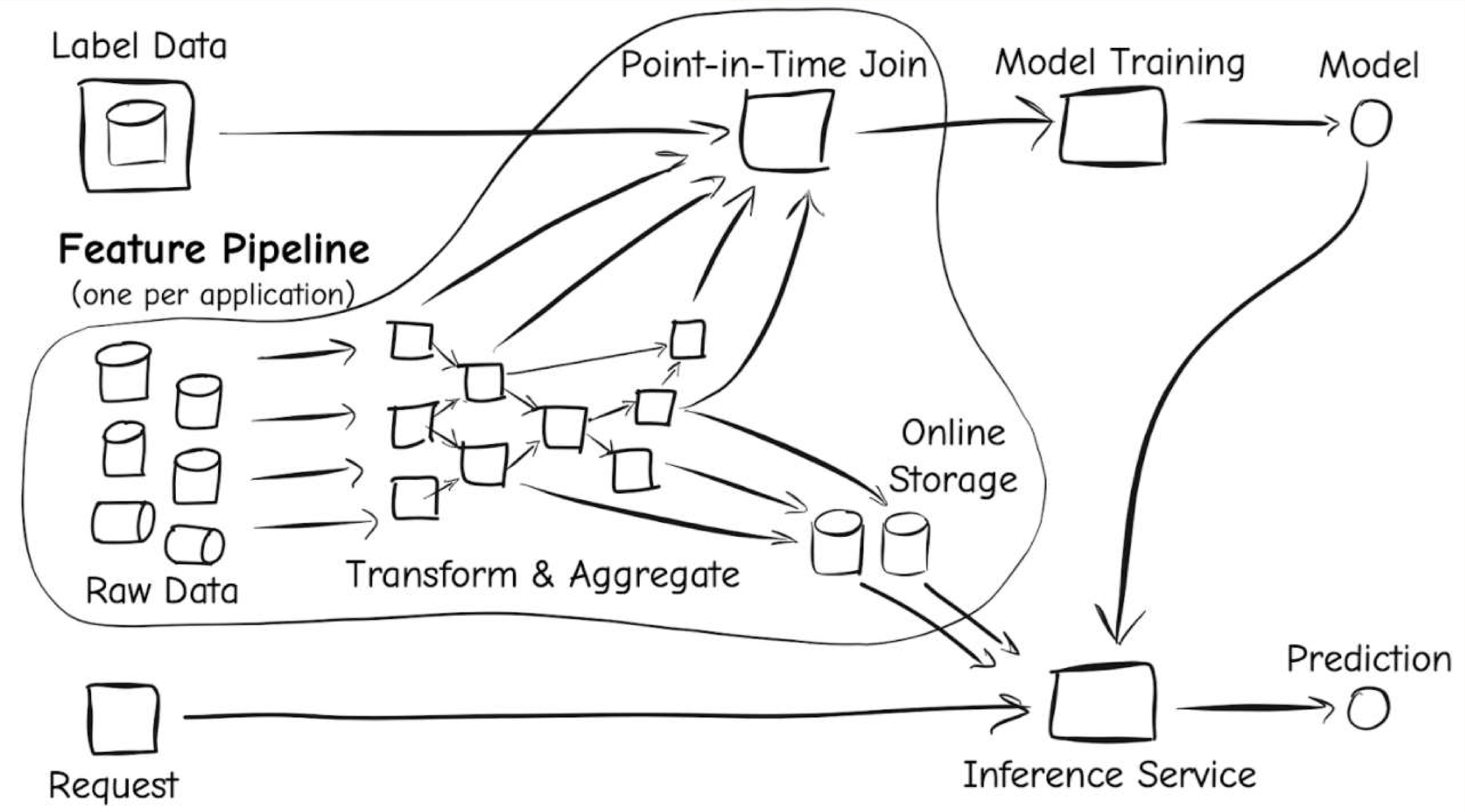
- Write jobs to get entity data from various sources
- Extract, aggregate, join, convert into proper format
- Load into model framework (different for train/serving)
- Worry about scale, perf, leakage, train/serve skew

New Way

- **Just import the feature by name into model code.**
- If feature doesn't exist, define and register it via simple APIs.

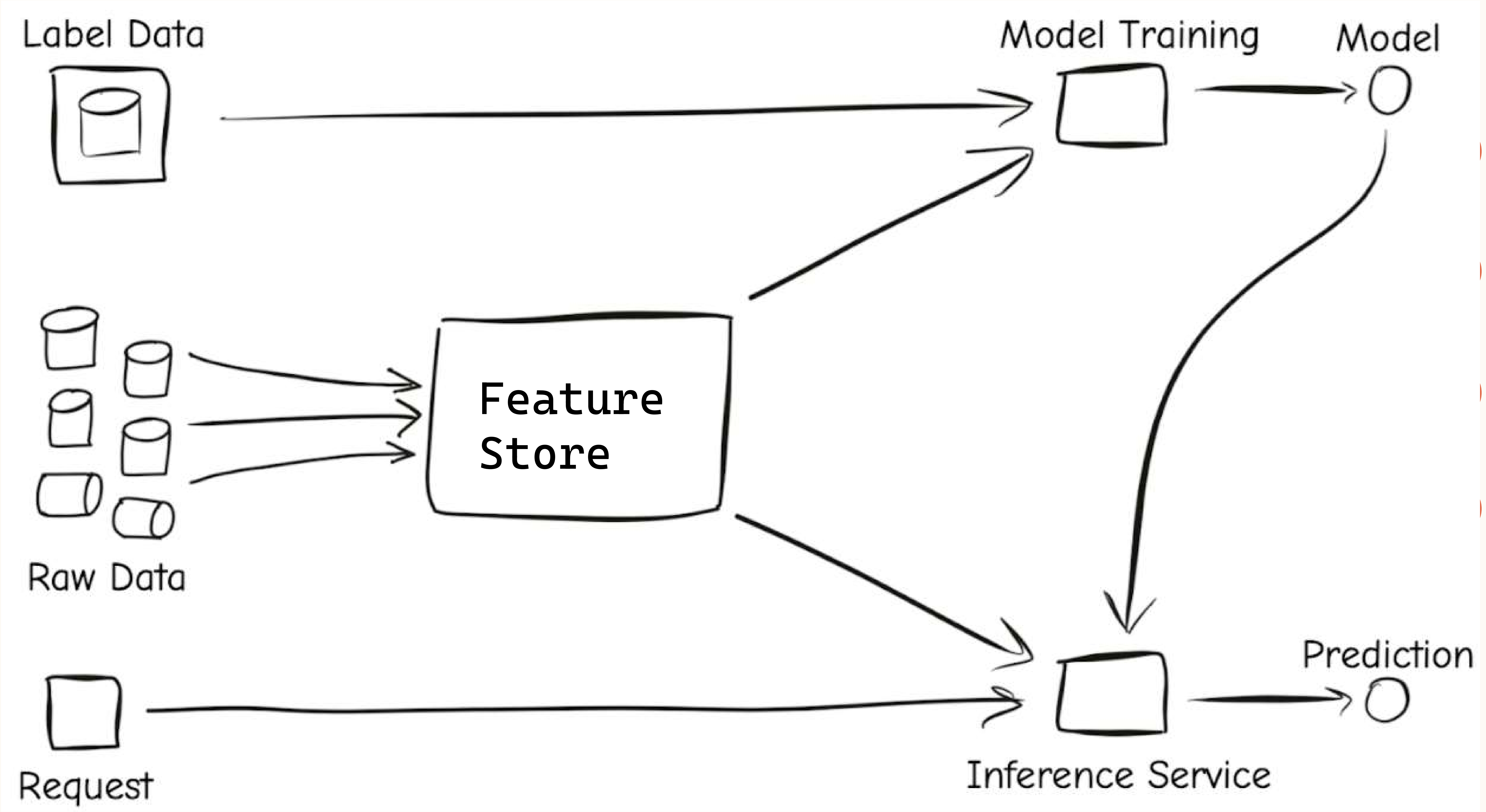
Why Feathr

Handle all the complex weight-lifting and “boring” work automatically



Why Feathr

Handle all the complex weight-lifting and “boring” work automatically



Like a package manager for feature engineering

Code

```
import module1
import module2
import module3
import module4
```

Features

```
query = FeatureQuery(
  feature_list=[
    "feature_1",
    "feature_2",
    "feature_3",
    "feature_4"
  ],
  key=item_id)
```

Problem: The complexity of feature preparation pipelines

1. Load & Transform

- Different programming APIs for different environments, e.g. online, offline, nearline, etc.
- Boilerplate and repetitive work
- Hard to test and debug

2. Train/Inference Skew

- Offline training and online inference usually require different data serving pipelines.
- Ensuring features are generated consistently is time intensive and error prone.
- Teams are deterred from using real time data for inferencing due to the difficulty in serving the right data.

3. Re-use and share

- The cost of building and maintaining feature pipelines was borne redundantly across many teams.
- Team-specific pipelines also made it impractical to reuse features across projects. e.g. no common type system, no common feature namespace

What a feature store should be

Feature store principal use cases

Develop Features

Based on raw data, using simple APIs

Deploy Features

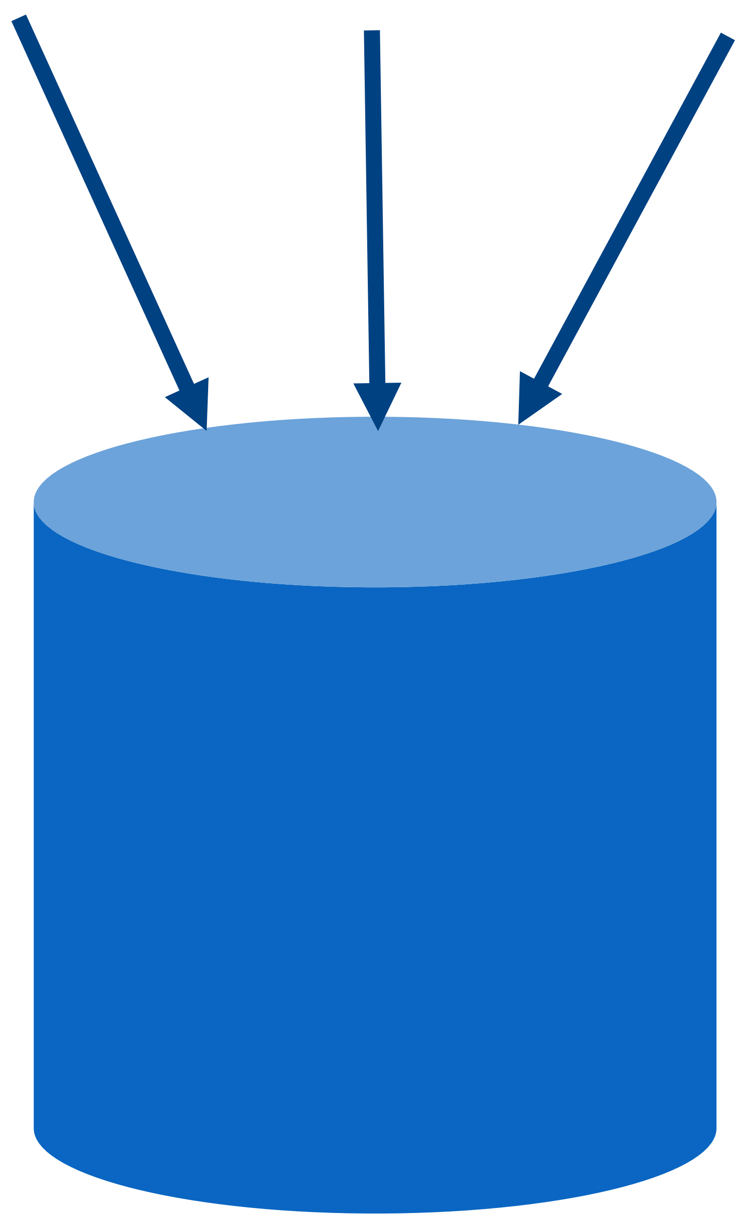
For training and online model inferencing

Manage Features

Monitor feature health and share across teams

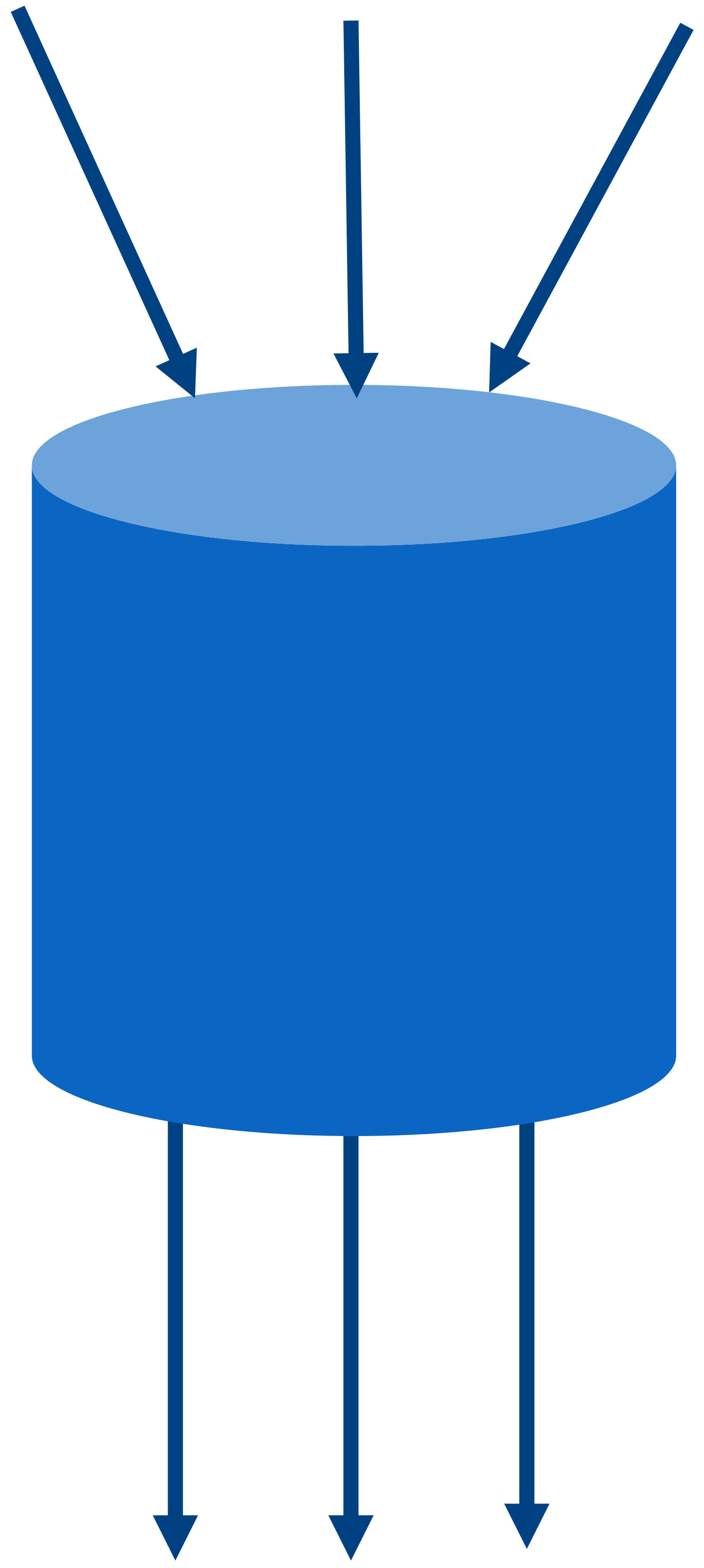
The “feature store” abstraction

- “Put a feature in” (Producer)
  - Develop a feature based on **raw data sets**
    - Sliding time windows
    - Aggregations
    - Transformations
    - Lookups/joins
  - Develop a feature based on other feature(s)



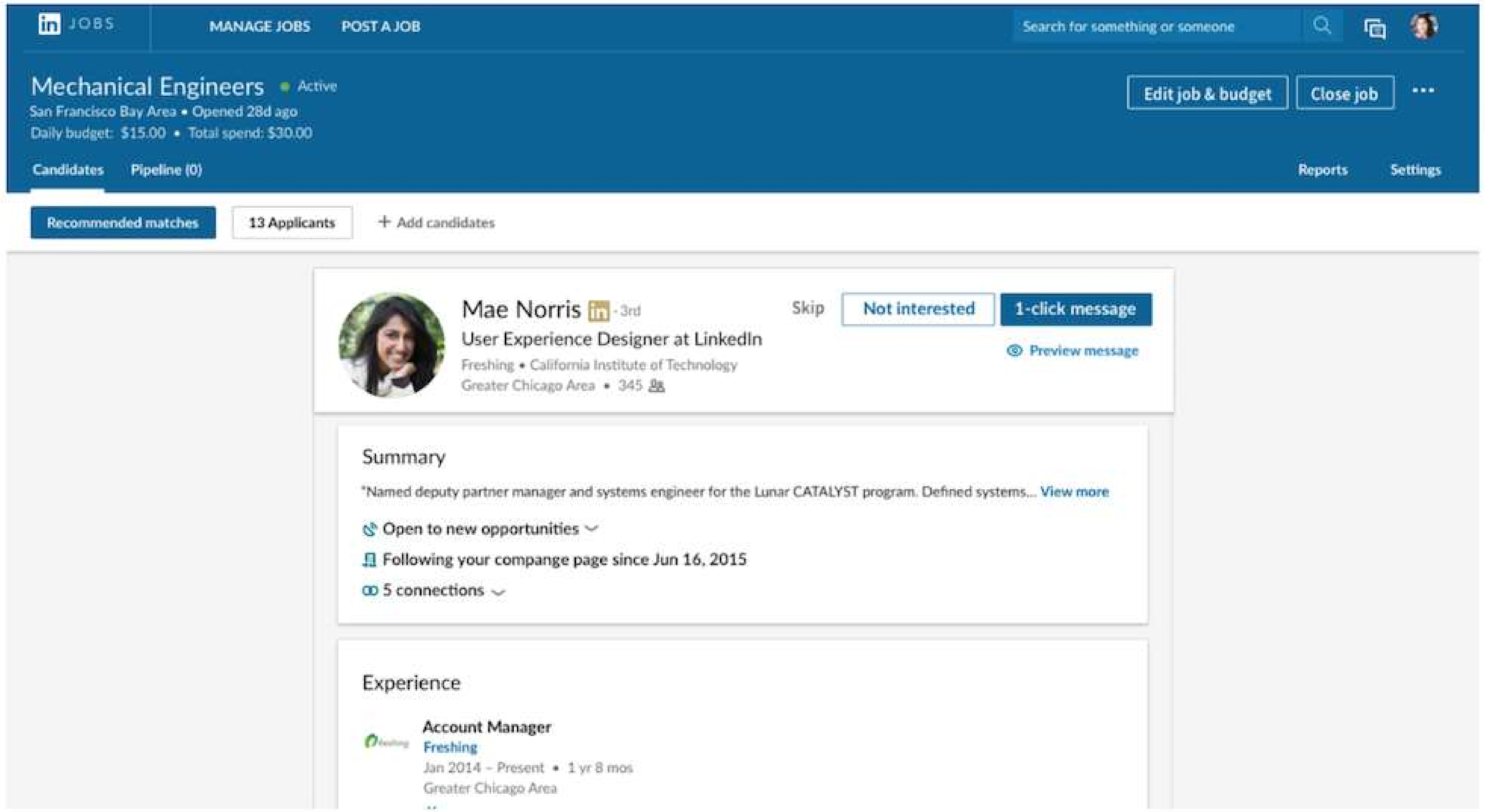
The “feature store” abstraction

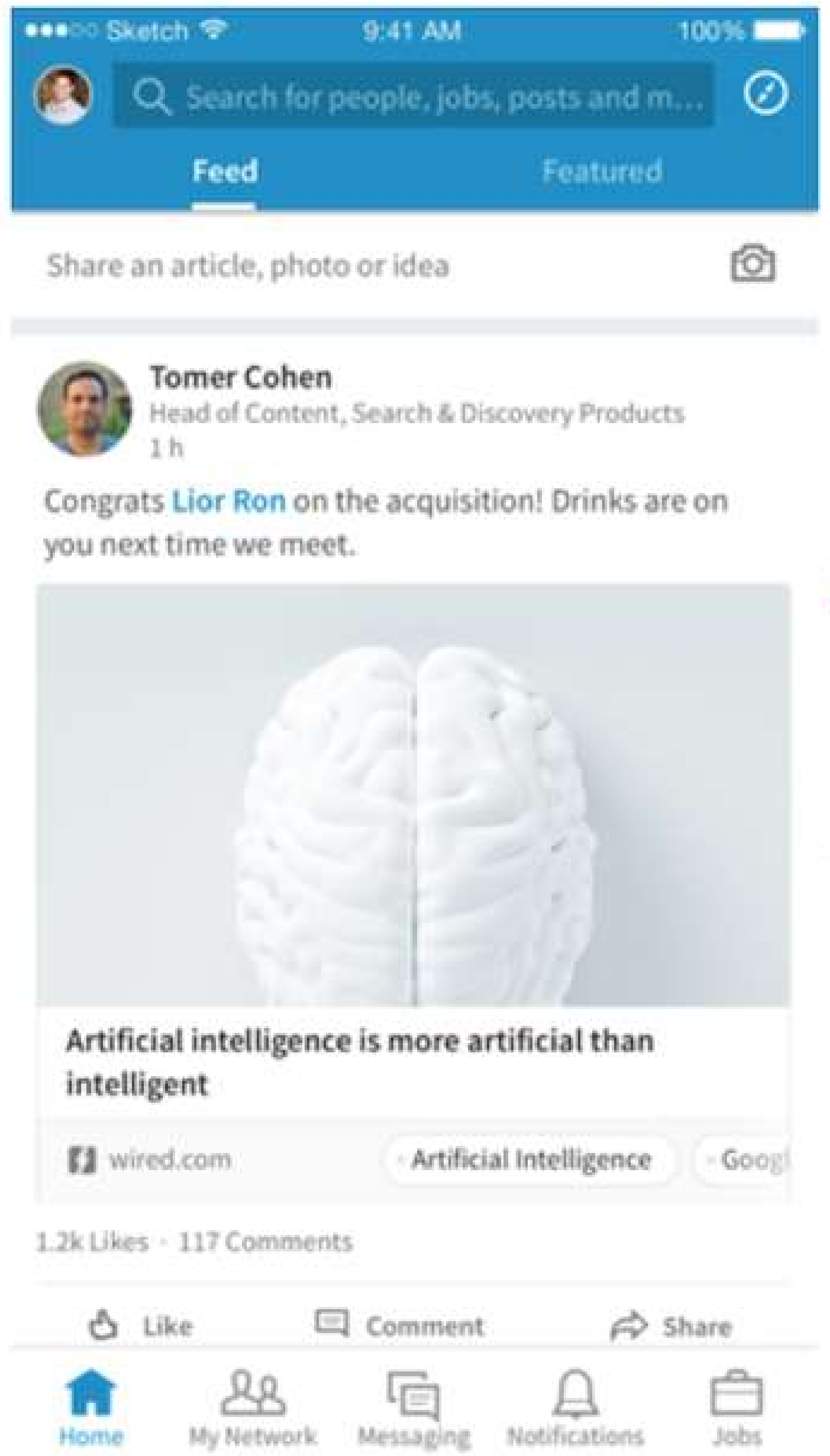
- “Put a feature in” (Producer)
  - Develop a feature based on **raw data sets**
    - Sliding time windows
    - Aggregations
    - Transformations
    - Lookups/joins
  - Develop a feature based on other feature(s)
- “Get some features out” (Consumer)
  - Join features to training labels
  - Backfill historical values of features (**point-in-time correctness**)
  - Efficiently compute, store, and serve features for **online inference**

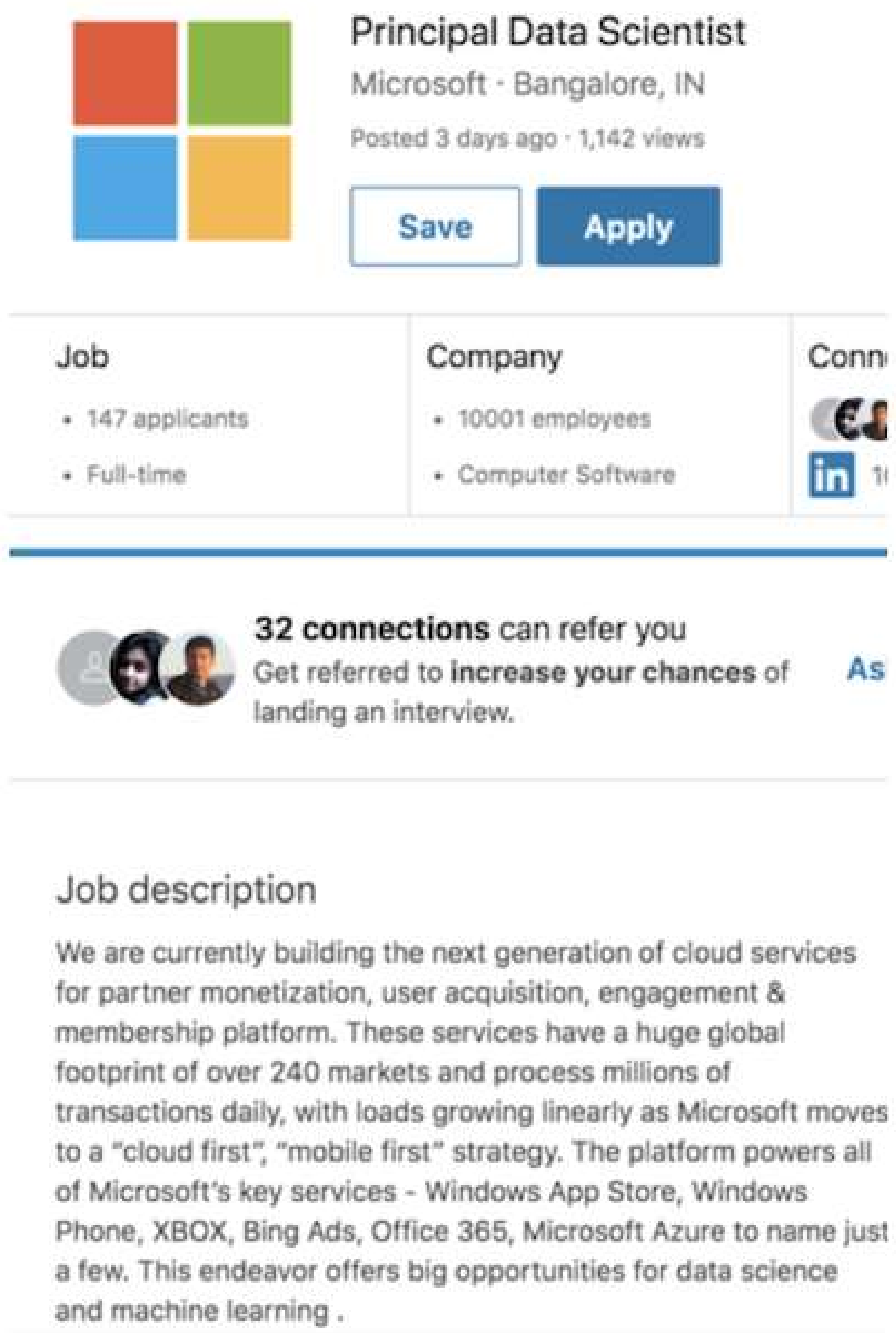


Feathr at LinkedIn

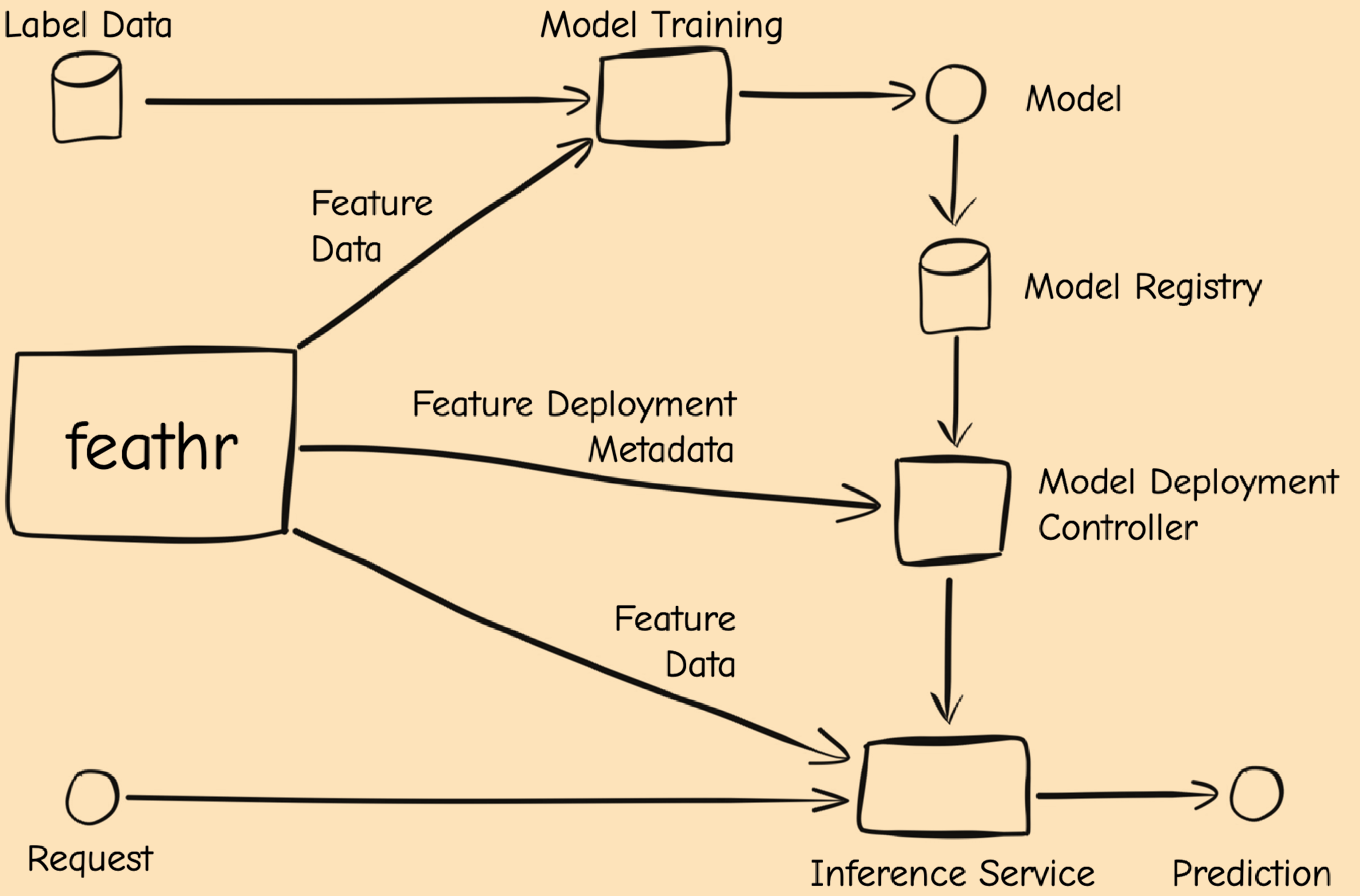
Introducing Feathr, a battle tested feature store built by LinkedIn







Learning Hiring Preferences: The AI Behind LinkedIn Jobs
Personalized Recommendations in LinkedIn Learning
Helping members connect to opportunity through AI
Near real-time features for near real-time personalization



Feathr is a pillar of LinkedIn's ML platform

Model deployment service uses Feathr to ensure a model's feature dependencies are deployed, before deploying the model.

Feathr at LinkedIn

- hundreds of models
- thousands of features
- many kinds of entities (economic graph)
- petabyte scale

Timeline

2017	Initial development and launch
2018	Broad adoption within LinkedIn
2020	Majority of LinkedIn ML applications onboarded
2022	Open source, Azure Integration, joined Linux Foundation AI & Data

Impact at LinkedIn

Majority of ML applications at LinkedIn have adopted Feathr

Improved Productivity

Faster experimentation with new features, from weeks to days

Improved Performance

Running time improved over custom pipelines, as much as 50%

Improved Collaboration

Applications can share features, which was difficult previously

What is Feathr

An abstraction layer between raw data and model

- **Define** features based on raw data sources using simple APIs.
- **Get** those features by their names during model training and model inferencing.
- **Share** features across your team and organizations.

Highlights

Cloud-native

Native integration with Azure and AWS

Rich Transformation

Python built-in transformations and PySpark UDF, on-demand evaluation

Scalable & High Performance

Highly optimized feature compute engine

Use case: Create Feature Definition

Load raw source data, and define transformation

```
batch_source = HdfsSource(
    name="nycTaxiBatchSource",                              # Source name to enrich your metadata
    path="abfss://green_tripdata_2020-04.csv",              # Path to your data
    event_timestamp_column="lpep_dropoff_datetime",         # Event timestamp for point-in-time correctness
    timestamp_format="yyyy-MM-dd HH:mm:ss")                 # Supports various fromats inculding epoch

trip_id = TypedKey(key_column="trip_id",
                   key_column_type=ValueType.INT64,
                   description="trip id")

features = [
    Feature(name="f_trip_distance",                         # Ingest feature data as-is
            feature_type=FLOAT,
            key=trip_id),
    Feature(name="f_is_long_trip_distance",
            feature_type=BOOLEAN,
            transform="cast_float(trip_distance)>30",
            key=trip_id)      # SQL-like syntax to transform raw data into feature
]

anchor = FeatureAnchor(name="anchor_features",              # Features anchored on same source
                       source=batch_source,
                       features=features)
```

Use Case - Streaming Feature

Create features from streaming source

```
stream_source = KafKaSource(name="kafkaStreamingSource",
                            kafkaConfig=KafkaConfig(brokers=["feathrazureci.servicebus.windows.net:
                                                    topics=["feathrcieventhub"],
                                                    schema=schema)
                            )

driver_id = TypedKey(key_column="driver_id",
                     key_column_type=ValueType.INT64,
                     description="driver id",
                     full_name="nyc driver id")

kafkaAnchor = FeatureAnchor(name="kafkaAnchor",
                            source=stream_source,
                            features=[Feature(name="f_modified_streaming_count",
                                              feature_type=INT32,
                                              transform="trips_today + 1",
                                              key=driver_id),
                                      Feature(name="f_modified_streaming_count2",
                                              feature_type=INT32,
                                              transform="trips_today + 2",
                                              key=driver_id)]
                            )
```

Use case: Build training dataset

Point-in-time Join Correct Semantics

```
# Requested features to be joined
# Define the key for your feature
location_id = TypedKey(key_column="DOLocationID",
                       key_column_type=ValueType.INT32,
                       description="location id in NYC",
                       full_name="nyc_taxi.location_id")
feature_query = FeatureQuery(feature_list=["f_location_avg_fare"], key=[location_id])

# Observation dataset settings
settings = ObservationSettings(
  observation_path="abfss://green_tripdata_2020-04.csv",    # Path to your observation data
  event_timestamp_column="lpep_dropoff_datetime",           # Event timepstamp field for your data,
  timestamp_format="yyyy-MM-dd HH:mm:ss")                   # Event timestamp format,  optional

# Prepare training data by joining features to the input (observation) data.
# feature-join.conf and features.conf are detected and used automatically.
feathr_client.get_offline_features(observation_settings=settings,
                                   output_path="abfss://output.avro",
                                   feature_query=feature_query)
```

Use Case - Feature Materialization

Materialize feature values to online storage for realtime access

```
client = FeathrClient()
redisSink = RedisSink(table_name="nycTaxiDemoFeature")
# Materialize two features into a redis table.
settings = MaterializationSettings("nycTaxiMaterializationJob",
sinks=[redisSink],
feature_names=["f_location_avg_fare", "f_location_max_fare"])
client.materialize_features(settings)
```

Use Case - Feature Sharing and Discovery

Share features and discover features

Use Case - Derived Feature

Define features on top of other features

```
# Compute a new feature(a.k.a. derived feature) on top of an existing feature
derived_feature = DerivedFeature(name="f_trip_time_distance",
                                 feature_type=FLOAT,
                                 key=trip_key,
                                 input_features=[f_trip_distance, f_trip_time_duration],
                                 transform="f_trip_distance * f_trip_time_duration")

# Another example to compute embedding similarity
user_embedding = Feature(name="user_embedding", feature_type=DENSE_VECTOR, key=user_key)
item_embedding = Feature(name="item_embedding", feature_type=DENSE_VECTOR, key=item_key)

user_item_similarity = DerivedFeature(name="user_item_similarity",
                                      feature_type=FLOAT,
                                      key=[user_key, item_key],
                                      input_features=[user_embedding, item_embedding],
                                      transform="cosine_similarity(user_embedding, item_embedding)"
```

Feathr Highlights – Scalability

- Capable of processing tens of billions of rows and PB scale data
- Native optimizations like bloom filters, join plan optimizer, salted join
- Incremental joins for large dataset

Feathr Architecture


Feathr UI (HA built-in with webapp)
Feathr REST API (HA built-in with webapp)
Access control
feature metadata
Feathr Feature Registry:
- Apache Atlas (Azure Purview)
- SQL Databases
Feathr Python Client
Feathr Python Client
Python APIs
Model Training:
Machine Learning Platform
- Azure Machine Learning
- Jupyter Notebook
- Databricks
Get online features
Feathr Offline Store:
Object Storage/HDFS
- Azure BLOB Storage
- Azure Data Lake Storage
- S3
- Delta Lake
feature materialization
Feathr Ingestion and Compute Engine: Apache Spark
- Databricks
- Azure Synapse
Feathr Online Store:
- Redis (Azure Redis Cache)
- Cosmos DB
Model Deployment
Feathr Offline Store:
SQL Database/Data Warehouse
- MySQL
- SQL Server
- Snowflake
Get offline features with:
- Point-in-time join correctness
- Multiple sources at once
Python APIs
Model Inference
Streaming Sources
- EventHub
- Kafka
Feathr also support Azure Key Vault to store credentials in a secure way.
Model Inference
Machine Learning Platform
- Azure Machine Learning
- Kubernetes
Streaming Features from a Stream Source to Online Store (using Spark)
Data Flow
Metadata Flow

Demo and Q&A

More Resources

Source code – welcome to start & fork!
https://github.com/feathr-ai/feathr

Tutorials:
Introduction to Feathr - Beginner's guide
Notebook tutorial: Build a Product Recommendation Machine Learning Model with Feathr Feature Store

Slack invitation:
https://join.slack.com/t/feathrai/shared_invite/zt-1ffva5u6v-voq0Us7bbKAw873cEzHOSg

Summary

- **Feathr** is an open-source feature store which can be seen as an abstraction layer between raw data and model.
- **Feathr** allows users to define features with transformation on top of raw data source and get feature values by feature name during both training and inferencing.
- **Feathr** simplifies feature preparation workflows and enables feature sharing across teams and company.

Thank you

(Check out our GitHub: https://github.com/feathr-ai/feathr)