Topic 6: ML Platforms and Feature Stores
ML Platforms in the Lifecycle

Data Scientist/ML Engineer

Source → Build → Deploy

ML/AI + Data Systems Infrastructure

- Data acquisition
- Data preparation
- Feature Engineering
- Training & Inference
- Model Selection
- Serving
- Monitoring
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.
Outline

❖ Technical Debt in Real-World ML
❖ The Rise of ML Platforms
  ❖ TensorFlow Extended
  ❖ MLFlow
❖ Feature Stores
Technical Debt in ML Applications

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Q: What sort of “debts” arise in real-world ML and why?
Q: What sort of “debts” arise in real-world ML and why?

- “Debt”: Creeping complexity in software management
  - Insidious errors/failures; reduces productivity; raises costs
- ML faces new (vs deterministic software) sources of debt:
  1. Datasets
  2. Data prep/featurization codes
  3. Auxiliary configurations
  4. Cross-tool/PL dependencies
  5. Concept evolution
Debt Source 1: Datasets

- ML model is a program derived from data, tested on data
- This causes 4 main types of debts:
  1. Lack of control over data sources/creators
  2. Inherent entanglement of features
  3. Lack of control over prediction consumers
  4. Scale-related errors
- Potential mitigation approaches:
  1. Mutually intelligible versioning of data and ML
  2. Refactor sources by importance
  3. Authentication and control of API usage
  4. Test for deployment scale
Debt Source 2: Data prep/featurization

- Raw data is seldom used as such in ML
- This causes 3 main types of debts:
  1. Pipeline jungles
  2. Unstable/inconsistent prep/featurization
  3. Wasteful retention of non-useful features
- Potential mitigation approaches:
  1. API-based prep/featurization frameworks
  2. Versioning of prep/featurization codes
  3. Refactor feature use by importance
Debt Source 3: Auxiliary configs

- Configs are ubiquitous in ML (code, training, prediction)
- This causes 3 main types of debts:
  1. Manual config management; forgotten configs
  2. Wrong config values; silent failures
  3. Dead experimentation codepaths
- Potential mitigation approaches:
  1. API-based config management
  2. Automated tests/analyses to vet configs
  3. Automated experimentation/metadata handling
End-to-end ML workflows often straddle multiple tools/PLs

- This causes 2 main types of debts:
  1. Glue code
  2. Abstraction debt

- Potential mitigation approaches:
  1. Encapsulation with multiple levels of APIs
  2. Standardize execution engines/frameworks
Debt Source 5: Concept Evolution

- ML is coupled to application’s underlying concept/distribution
- This causes 2 main types of debts:
  1. Concept drift
  2. Feedback loops
- Potential mitigation approaches:
  1. Monitoring and actionable recourse
  2. Organization-wide A/B test controls
Discussion on Technical Debt paper and Martin’s Rules of ML Engineering
Outline

- Technical Debt in Real-World ML
- The Rise of ML Platforms
  - TensorFlow Extended
  - MLFlow
- Feature Stores
“Platformization” of ML is not new; 30+ years of work!

SAS; In-RDBMS ML work fine in many enterprise settings
The Original ML “Platform”

- JMP (1989) other tools offered GUI + IDE for R-like PL
- In-house governed environment with data querying, viz, ML, versioning, deployment, etc.
- SAS still has largest market share in enterprise ML!

[Links]
Q: So what changed? Why the need for new ML platforms?

- 6 “revolutions” in CS in the last 20yrs:
  - Web; IoT; Cloud; “Big Data”; Open source; DL
- Rapid growth in complexity of ML landscape:
  - **Heterogeneity** of ML apps, deployment regimes, and user base
  - Data *scale, variety, and velocity*
  - **Open source tool boom** for data and ML
  - **Flexibility** and empiricism of DL
- Created new jobs: “Data Scientist” / “ML Engineer”
Introducing ML Platforms

❖ **Goal:** Pay off ML technical debt for *modern ML*
❖ No consensus yet on “canonical” architecture but 3 things:
  ❖ **End-to-end governance:** access control, versioning, usage control, deployment support, monitoring
   ❖ **Share components** across apps/users; less glue code, more code reuse, more automation
   ❖ **Standardization of APIs** for data, features, ML code, configs; lower abstraction debt, fewer pipeline jungles, less manual handling/errors
   ❖ ML platforms : ML systems :: RDBMSs : Scalable SQL?
Outline

❖ Technical Debt in Real-World ML
❖ The Rise of ML Platforms
❖ TensorFlow Extended
   ❖ MLFlow
❖ Feature Stores
TensorFlow Extended (TFX)

- Pioneering ML platform from Google
- Built around TF, although many parts are not TF-specific

https://www.tensorflow.org/tfx
Goal: Standardize end-to-end ML pipelines, simplify component use, and make production ML easier
TensorFlow Extended (TFX)

- System design philosophy of TFX:
  - **Loose coupling** for data ingestion
  - **Modularity** of platform components
  - **Standardized metadata** handling
  - **Standardized APIs** to call out training/serving; for prep
  - **Automation** of validation, config handling
  - **Native support** for monitoring, debugging
Key current parts of TFX:

- **ML Metadata**: log execution lineage, artifacts
- **TFDV**: validate data, handle features
- **TF Transform**: tensor proc. functions; Apache Beam
- **TF plugins**: ingest data, model; emit model
- **TF Model Analysis**: validate model, monitor metrics
- **TF Serving/Lite/JS**: deploy to appl. environment

https://www.tensorflow.org/tfx
TFX’s ML Metadata (MLMD)

Pipeline components/steps

MLMD client libraries

Metadata and lineage about input/output artifacts

ML Platform / Workflow (e.g. TFX)

Graphical User Interface

MetadataStore

Event: input

Event: output

Attribution

Context: ContextType

Artifact: ArtifactType

Execution: ExecutionType

Component of ML Metadata

SQLLite Source

MySQL Source

...
TFX’s ML Metadata (MLMD)

1) Register ArtifactTypes
2) Register ExecutionTypes
3) Create DataSet Artifact
4) Create Execution for Trainer
5) Read DataSet and record input event
6) Train Model and Create SavedModel Artifact
7) Write SavedModel and record output event
8) Mark Execution completed
9) Annotate the experiment with a Context

https://www.tensorflow.org/tfx/guide/mlmd
TFX’s ML Metadata (MLMD)

❖ User must *register* metadata, artifacts, pipelines with code

1) Register artifact types

```
# Create ArtifactTypes, e.g., Data and Model
data_type = metadata_store_pb2.ArtifactType()
data_type.name = "DataSet"
data_type.properties["day"] = metadata_store_pb2.INT
data_type.properties["split"] = metadata_store_pb2.STRING
data_type_id = store.put_artifact_type(data_type)

model_type = metadata_store_pb2.ArtifactType()
model_type.name = "SavedModel"
model_type.properties["version"] = metadata_store_pb2.INT
model_type.properties["name"] = metadata_store_pb2.STRING
model_type_id = store.put_artifact_type(model_type)
```

2) Register execution types for all steps in the ML workflow

```
# Create an ExecutionType, e.g., Trainer
trainer_type = metadata_store_pb2.ExecutionType()
trainer_type.name = "Trainer"
trainer_type.properties["state"] = metadata_store_pb2.STRING
trainer_type_id = store.put_execution_type(trainer_type)
```

3) Create an artifact of DataSet ArtifactType

❖ **Pros:** Easier to track, helps governance, can automate post-hoc debugging,

❖ **Cons:** Friction for user, boilerplate overhead, manual errors

https://www.tensorflow.org/tfx/guide/mlmd
TFX’s Model Analysis (TFMA)

- Automated model quality metrics and plots; handle data splits; track over time
- Standardized handling of prediction thresholds
- “Fairness” Indicators to specify groups, catch divergence

https://www.tensorflow.org/tfx/guide/fairness_indicators
Outline

❖ Technical Debt in Real-World ML
❖ The Rise of ML Platforms
  ❖ TensorFlow Extended
  ❖ MLFlow
❖ Feature Stores
**Goal**: Standardize and manage end-to-end ML pipelines, simplify component use, and make production ML easier

- Key philosophical difference: user can bring *any* ML tool

**Just like TFX:**
- Open source, modular, loose coupling of data, handles metadata, has callouts to general training/serving/etc.

**Unlike TFX:**
- No standardizing of component internals; instead, basic connectors with CLIs and REST APIs
- In-built connectors for popular ML tools beyond TF
Key current parts and in-built connectors in MLFlow:

- **mlflow TRACKING**
  - Experiment and metric tracking

- **mlflow PROJECTS**
  - Reproducible execution

- **mlflow MODELS**
  - Model packaging and deployment

- **mlflow MODEL REGISTRY**
  - Model management

- Built-in connectors

Parts differ quite a bit from TFX due to MLFlow’s emphasis on higher generality of ML tool support.
MLFlow’s Tracking

- API to log code version, source, time, parameters, metrics, and output files (models/data/logs)

```python
with mlflow.start_run():
    for epoch in range(0, 3):
        mlflow.log_metric(key="quality", value=2*epoch, step=epoch)
```

```java
MlflowClient client = new MlflowClient();
RunInfo run = client.createRun();
for (int epoch = 0; epoch < 3; epoch++) {
    client.logMetric(run.getRunId(), "quality", 2 * epoch, System.currentTimeMillis(), epoch);
}
```

- Automated logging for many popular ML tools
- GUI-based visualization of tracked content

https://mlflow.org/docs/latest/tracking.html
MLFlow’s Projects

- API to **package** ML experiment for reusability/reproducibility
- Record environment or even container with YAML config file
- Can version, authenticate, re-deploy projects more easily

```yaml
name: My Project

conda_env: my_env.yaml
# Can have a docker_env instead of a conda_env, e.g.
# docker_env:
#   image: mlflow-docker-example

entry_points:
    main:
        parameters:
            data_file: path
            regularization: {type: float, default: 0.1}
        command: "python train.py -r {regularization} {data_file}"
    validate:
        parameters:
            data_file: path
        command: "python validate.py {data_file}"```
MLFlow’s Models

❖ General meta-format to save and exchange ML models
❖ Easier to archive, govern, and connect with serving
❖ Spectrum of support for how much to “look into” model; standard “flavors” for popular tools; new tools more generic

```python
# Directory written by mlflow.sklearn.save_model(model, "my_model")
my_model/
  MLmodel
  model.pkl
```

And its `MLmodel` file describes two flavors:

```yaml

flavors:
sklearn:
  sklearn_version: 0.19.1
  pickled_model: model.pkl
python_function:
  loader_module: mlflow.sklearn
```

https://mlflow.org/docs/latest/models.html
MLFlow’s Models

▫ Model “Signature” to enforce data schemas; JSON structs
▫ Similar to TFX’s MLMD; same pros/cons on usage

```python
import pandas as pd
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
import mlflow
import mlflow.sklearn
from mlflow.models.signature import infer_signature

iris = datasets.load_iris()
iris_train = pd.DataFrame(iris.data, columns=iris.feature_names)
clf = RandomForestClassifier(max_depth=7, random_state=0)
clf.fit(iris_train, iris.target)
signature = infer_signature(iris_train, clf.predict(iris_train))
mlflow.sklearn.log_model(clf, "iris_rf", signature=signature)
```

https://mlflow.org/docs/latest/models.html
MLFlow’s Model Registry

- Governed facility for storing and shipping MLFlow Models
- Lineage, versioning, usage control; listing and searching
- Easier to connect with serving infra.

Registered Models > Risk Model > Version 1

Registered At: 2020-03-08 11:30:28
Creator: Corey
Stage: Production
Last Modified: 2020-03-08 11:46:21
Source Run: Run e7e695f63a334259b665f789b5da84b0

Description

ElasticNet regression model trained on credit approval dataset

```python
create_registered_model(
    name="Risk Model"
)
create_model_version(
    name="Risk Model",
    source="/.../riskmodel"
)
transition_model_version_stage(
    name="Risk Model",
    version=1,
    stage="Staging"
)
load_model(
    "models:/Risk Model/Staging"
)
```

https://mlflow.org/docs/latest/model-registry.html
Comparing TFX vs MLFlow

❖ Adoption:
  ❖ TFX: widely used in Google; some Web firms
  ❖ MLFlow: popular in enterprises, some Web, domain sci.

❖ Pros/cons of TFX’s approach vs MLFlow’s approach largely same as what we saw for the 2 types of ML serving systems
  ❖ But: Many of TFX’s parts are not TF-specific
  ❖ But: MLFlow is specializing support for a few popular ML tools (e.g., PyTorch, XGBoost)

Q: What ML technical debts do these tools resolve vs not?
Discussion on paying off ML technical debt with TFX vs MLFlow
Rate the paper's innovativeness/novelty/creativity
37 responses

Rate the paper's technical depth
37 responses
Rate the paper's innovativeness/novelty/creativity
38 responses

Rate the paper's technical depth
38 responses
Outline

❖ Technical Debt in Real-World ML
❖ The Rise of ML Platforms
   ❖ TensorFlow Extended
   ❖ MLFlow
❖ Feature Stores
Introducing Feature Stores

❖ **Goal:** Organize feature engineering outputs and reduce feature-related technical debt in ML

❖ No consensus on definition yet but one attempt:

❖ Feature store is an ML-specific data system to:
  ❖ Run **data pipelines that transform** raw data into feature values
  ❖ **Stores and manages** the feature data itself
  ❖ **Serves feature data** consistently for training and inference purposes

https://www.tecton.ai/blog/what-is-a-feature-store/
Introducing Feature Stores

- No consensus yet on canonical architecture but typically:

https://www.tecton.ai/blog/what-is-a-feature-store/
Introducing Feature Stores

❖ No consensus yet on canonical architecture but 3 key parts:
❖ **Data Warehouse for Features**: Ingestion, storage, governance (access/usage control, sharing, versioning) of prep/featurization codes and/or *materialized* features
❖ **Feature Serving**: Deliver featurization code and/or materialized features to online prediction serving
❖ **Standardization of APIs** for feature engineering, integration with online prediction serving; fewer pipeline jungles, less manual handling/errors
❖ Typically built on top of a DB/dataflow system + K-V store

https://www.tecton.ai/blog/what-is-a-feature-store/
https://medium.com/data-for-ai/what-is-a-feature-store-for-ml-29b62580af5d
Benefits of Feature Stores

- Abstracting out and refactoring featurization enables **consistency** across Build and Deploy stages
  - Often distinguishes as Offline vs Online Serving
  - Offline typically feeds features to building/training
  - Online typically feeds features to online prediction serving
- **Sharing and reuse** of feature code can improve productivity
- **Automation** of feature serving, monitoring, versioning

[https://www.tecton.ai/blog/what-is-a-feature-store/](https://www.tecton.ai/blog/what-is-a-feature-store/)
[https://medium.com/data-for-ai/what-is-a-feature-store-for-ml-29b62580af5d](https://medium.com/data-for-ai/what-is-a-feature-store-for-ml-29b62580af5d)
Online vs Offline Feature Serving

https://medium.com/data-for-ai/what-is-a-feature-store-for-ml-29b62580af5d
Data Warehouse vs Feature Store

- Feature store (FS) can be seen as a specialized data warehouse tailored for ML workloads + extra capabilities
- **Target users:** DW mainly for data analysts; FS for ML users
- **Purpose of Data:** DW is primarily for (semi)structured data with schema; FS must support all ML feature types
- **Metadata:** Both manage and maintain this
- **Offline workloads:** DW is full SQL; FS is a custom API with complex set of ops subsuming SQL, UDFs, etc.
- **Online workloads:** DW is full SQL; FS typically offers K-V lookups; some joins, filters

Users/Workloads for a Feature Store

https://www.tecton.ai/blog/what-is-a-feature-store/
SignalHub (c. ~2016)

- One of the earliest enterprise feature stores
- No support for Online
- Mainly used in-house

Tecton (2020)

https://www.tecton.ai/blog/what-is-a-feature-store/
Tecton: Overall Architecture

https://www.tecton.ai/blog/what-is-a-feature-store/
Tecton: Storage

https://www.tecton.ai/blog/what-is-a-feature-store/
Tecton: Serving

Request feature vector for inference

.feature_vector (  
features=["item_clicks_17d",  
"item_price",  
"user_spend_130d"],  
entities={"user_id":3,  
"item_id":62}  
)

Feature Vector  
(latest values, low-latency)

Rest API

Request training data

.get_training_data (  
features=["item_clicks_17d",  
"item_price",  
"user_spend_130d"],  
entities=entities_df  
)

Training Dataframe  
(historical values)

<table>
<thead>
<tr>
<th>user_id</th>
<th>item_id</th>
<th>time</th>
<th>item_clicks_17d</th>
<th>item_price</th>
<th>user_spend_130d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>54</td>
<td>7485</td>
<td>6</td>
<td>$23.2</td>
<td>$0</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>46854</td>
<td>5</td>
<td>$41.54</td>
<td>$8.41</td>
</tr>
</tbody>
</table>

https://www.tecton.ai/blog/what-is-a-feature-store/
Tecton: Transformation

Feature Store

Transforms
- Stream transform
- Batch transform

Storage
- Precomputed values

Serving
- On-demand transform

Application data
e.g. `{user_ip: "x.x.x.x"}`

Feature Vector

```
{
  "item_clicks_last_5_mins": 6,
  "user_current_location": "US_NY",
  "user_spend_last_30_days": "$1.16"
}
```

https://www.tecton.ai/blog/what-is-a-feature-store/
## Tecton: Transformation Taxonomy

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Definition</th>
<th>Common input data source</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Transform</td>
<td>Transformations that are applied only to data at rest</td>
<td>Data warehouse, data lake, database</td>
<td>User country, product category</td>
</tr>
<tr>
<td>Streaming Transform</td>
<td>Transformations that are applied to streaming sources</td>
<td>Kafka, Kinesis, PubSub</td>
<td># of clicks per vertical per user in last 30 minutes, # of views per listing in past hour</td>
</tr>
<tr>
<td>On-demand transform</td>
<td>Transformations that are used to produce features based on data that is only available at the time of the prediction. These features cannot be pre-computed.</td>
<td>User-facing application</td>
<td>Is the user currently in a supported location? Similarity score between listing and search query</td>
</tr>
</tbody>
</table>

[https://www.tecton.ai/blog/what-is-a-feature-store/](https://www.tecton.ai/blog/what-is-a-feature-store/)
### Comparing Modern Feature Stores

<table>
<thead>
<tr>
<th>Platform</th>
<th>Open-Source</th>
<th>Offline</th>
<th>Online</th>
<th>Metadata</th>
<th>Feature Engineering</th>
<th>Supported Platforms</th>
<th>TimeTravel / Point-in-Time Queries</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hopsworks</strong></td>
<td>AGPL-V3</td>
<td>Hudi/Hive</td>
<td>MySQL Cluster</td>
<td>DB Tables, Elasticsearch</td>
<td>(Py)Spark, Python</td>
<td>AWS, GCP, On-Prem</td>
<td>SQL Join or Hudi Queries</td>
<td>.tfrecords, .csv, .npy, .petastorm, .hf5, etc</td>
</tr>
<tr>
<td><strong>Michelangelo</strong></td>
<td>N/A</td>
<td>Hive</td>
<td>Cassandra</td>
<td>Content</td>
<td>Spark, DSL</td>
<td>Proprietary</td>
<td>SQL Join</td>
<td>Streamed to models?</td>
</tr>
<tr>
<td><strong>Feast</strong></td>
<td>Apache V2</td>
<td>BigQuery</td>
<td>BigTable/Redis</td>
<td>DB Tables</td>
<td>Beam, Python</td>
<td>GCP</td>
<td>SQL Join</td>
<td>Streamed to models</td>
</tr>
<tr>
<td><strong>Conde Nast</strong></td>
<td>N/A</td>
<td>Kafka/Cassandra</td>
<td>Kafka/Cassandra</td>
<td>Protocol Buffers</td>
<td>Shared libraries</td>
<td>Proprietary</td>
<td>?</td>
<td>Protobuf</td>
</tr>
<tr>
<td><strong>Zipline</strong></td>
<td>N/A</td>
<td>Hive</td>
<td>KV Store</td>
<td>KV Entries</td>
<td>Flink, Spark, DSL</td>
<td>Proprietary</td>
<td>Schema</td>
<td>Streamed to models?</td>
</tr>
</tbody>
</table>

[https://www.featurestore.org/](https://www.featurestore.org/)
Outline

❖ Technical Debt in Real-World ML
❖ The Rise of ML Platforms
  ❖ TensorFlow Extended
  ❖ MLFlow
❖ Feature Stores
Open Research Questions

- ML platforms and features stores are fast evolving; canonical architectures are starting to form
- How to evaluate expressiveness of such platforms?
- How to benchmark their resource efficiency and costs?
- More holistic end-to-end pipelines optimizations
- Applying program analysis to automate optimizations
- Debugging, provenance management, “what if” questions