Architecting to Support Machine Learning

Humberto Cervantes, UAM
Iurii Milovanov, SoftServe
Rick Kazman, University of Hawaii
PARTICULARITIES OF ML SYSTEMS

- In ML systems, the behaviour is not specified directly in code but is learned from data.

- At the core of the system, there is a model that uses data transformed into features to perform predictions for particular tasks.
TWO MAIN WORKFLOWS

Development environment

Raw historical data

Transformation into features

Model selection and training

Trained ML Model

Model development

model development

data transformation rules + model

data to refine model & data rules

New raw data

Transformation into features

Trained ML Model

Results derived from prediction

model serving

Serving environment

automatic retraining
The development of ML systems frequently follows a sequential approach.
ML SYSTEM DEVELOPMENT

But something closer to this is needed...

Initial Model development

Model serving

Model refinement

(Refined) Model Serving

Model refinement

(Refined) Model Serving
ARCHITECTING THE SYSTEM

Supporting these aspects

Introduces many architectural concerns:

“Architectural concerns encompass additional aspects that need to be considered as part of architectural design but which are not expressed as traditional requirements.”
We will look into more details in the steps of the workflows to discuss the concerns and decisions that can be made to satisfy them.
Responsibility

● Collect and store raw data for training

Architectural concerns

● Collect and store large volumes of training data, support fast bulk reading
  ○ Ingestion: Manual, Message broker, ETL Jobs
  ○ Storage: Object Storage, SQL or NoSQL, HDFS
● Labeling of raw training data
  ○ Data labelling toolkit: Intel’s CVAT, Amazon Sagemaker Ground Truth
● Protect sensitive data
DATA CLEANSING AND NORMALIZATION

Responsibility

- Identify and remove errors and duplicates from selected data and perform data conversions (such as normalization) to create a reliable data set.

Architectural concerns

- Provide mechanisms such as APIs to support query and visualization of the data
  - Data warehouse to support data analysis, such as HIVE
- Transform large volumes of raw training data
  - Data processing framework, such as Spark
FEATURE ENGINEERING

Responsibility
- Perform data transformations and augmentation to incorporate additional knowledge to the training data
- Identify the list of features to use for training

Architectural concerns
- Transform large volumes of raw training data into features
- Provide mechanism for data segregation (training / testing)
- Features logging and versioning
  - Logging mechanism, such as Stackdriver Logging
  - Data versioning mechanism, such as Data Science Version Control System (DVC)
MODEL TRAINING AND SELECTION

Responsibility

- Based on a selected algorithm, train, tune and evaluate a model.

Architectural concerns

- Selection of a framework
  - TensorFlow, PyTorch, Spark MLlib, scikit-learn, etc.
- Select training location and provide environment and manage resources to train, tune and evaluate a model
  - Single vs distributed training, Hardware acceleration (GPU/TPU)
  - Resource Management (e.g. Yarn, Kubernetes)
- Log and monitor training performance metrics
Responsibility

- Persist the trained and tuned model (or entire pipeline) to support transfer to the serving environment

Architectural concerns

- Persistence of the model
  - Examples: Spark MLlib Pipelines, PMML, MLeap, ONNX
- Storage of the model
  - Examples: Database, document storage, object storage, NFS, DVC
- Optimize model after training (e.g. reduce size for use in constrained device)
  - Example: Tensorflow Model Optimization Toolkit
NEW DATA INGESTION

Responsibility

- Obtain and import unseen data for predictions

Architectural concerns

- Batch prediction: asynchronously generate predictions for multiple input data observations.
- Online (or real-time) prediction: synchronously generate predictions for individual data observations.
DATA VALIDATION AND FEATURE EXTRACTION

Responsibility

● Process raw data into features according to the transformation rules defined during model development

Architectural concerns

● Ensure data conforms to the rules defined during training
  ○ Usage of a data schema defined during model development

● Design batch and/or streaming pipelines
  ○ Realtime data storage (e.g. Cassandra)
  ○ Data processing framework (e.g. Spark)

● Select and query additional real-time data sources (if needed)
MODEL TRANSFER AND PREDICTION

Responsibility
● Transfer of model code and perform predictions

Architectural concerns
● Define prediction location
● Model transfer and validation
  ○ Transfer: re-writing, docker, PMML...
  ○ Support for multiple model versions, update and rollback mechanisms, for example using TensorFlow serving
**Prediction Location**

Local model: the model predicts/re-trains on the client side

Remote model: the model predicts/re-trains on the server side

Hybrid model predicts on client and re-trains on both (federated learning)
SERVING RESULTS

Responsibility

● Monitoring and delivery of prediction results to a destination

Architectural Concerns

● Monitor model staleness (age) and performance
● Monitoring deviations between distribution of predicted and observed labels
● Canary and A/B testing
● Storage prediction results
● Aggregation results from multiple models
CASE STUDIES
NEW DOMAIN UNDERSTANDING
• SoftServe worked with two Fortune 100 companies – an IT, hardware and networking provider, and an energy exploration and production company – to research the oil extraction process
• SoftServe suggested a solution and architecture design to match the client need for a distributed fiber-optic sensing (IoT) program.

DOMAIN-SPECIFIC TECHNOLOGY CHALLENGES / LIMITATIONS
• SoftServe suggested 3rd-party sensing hardware (Silixa) and data protocol (National Instruments) to address industry-specifics challenges
• SoftServe designed and deployed a hybrid edge and cloud data processing model
• We built a real-time BI layer and analytics engine on large-scale data streams

SOLUTION DESIGN
• SoftServe’s end solution focused on unsupervised anomaly detection to help the end client identify observations that do not conform to the expected behavioral patterns
ARCHITECTURAL DRIVERS

• Ingest and process multi-dimensional time series streaming data from sensors (100-200GB per day).
• Calculate the key metrics and perform short- and long-term predictions over different historical windows in near real-time (up to 5 mins)
• The model should be able to continuously re-train when the new data comes in
• Initial training dataset consisted of ~300GB
• Support queries against historical data for analytics
ARCHITECTURAL DECISION [MODEL DEV]

**Training Data Ingestion**
- HDFS used as a storage layer
- Directory structure for data versioning
- Custom data conversion from the proprietary data protocol

**Feature engineering**
- Batch Spark job to calculate the features
- Selected features were stored in CrateDB and exposed via SQL

**Data cleansing and normalization**
- Spark SQL and Dataframes for analytics
- Batch Spark jobs for data pre-processing

**Model training and selection**
- Spark ML for model training and tuning
- Yarn resource management
- No hardware acceleration were used

**Model persistence**
- The result models were stored on HDFS
ARCHITECTURAL DECISION [MODEL SERVING]

New Data Ingestion
- Kafka used as a message broker to ingest the data from the sensors

Data validation an Feature extraction
- Same batch transformations re-used in Spark Streaming

Model prediction
- Batch Spark ML jobs scheduled every 3 mins

Serving results
- The results saved back to CrateDB and exposed via Impala
- Zoomdata used to communicate the data and predictions
CASE STUDY
DISTRIBUTED IOT NETWORK ACROSS OIL & GAS PRODUCTION

SOFTWARE SOLUTION TEAM

- Technical lead (Java, Scala, Hadoop, Spark)
- Big Data architect (Cloudera, Hadoop, Spark, Kafka, CrateDB, Impala)
- Senior backend engineers (Java, Scala)
- Frontend engineers (JavaScript, Zoomdata)
- DevOps engineers (Ansible, Docker, Mesos)
- Data scientists (Machine Learning, DSP, time-series analytics)

![Diagram of IoT network architecture]

1. **Sensors**
2. **kafka**
3. **Streaming Layer**
   - Spark Streaming
   - StreamSets
4. **Storage Layer**
   - Hadoop
   - Hops
   - Crate
5. **Processing Layer**
   - Spark
   - SQL
6. **BI & Analytics**
   - Cloudera
   - Impala
   - Zoomdata
OUTCOMES

• Highly scalable distributed IoT platform leveraging state-of-the-art Big Data and Cloud technologies
• Real-time monitoring and user-centric BI analytics
• Custom domain-specific self-learning anomaly detection solution
A SoftServe innovative solution provides automatic parking space detection based on a Computer Vision ML model.

A CCTV camera installed on a rooftop captures images and the current parking state is visualized in real-time via a web application and LCD at the parking entrance.

The solution can be used for both open and authorized parking areas.
ARCHITECTURAL DRIVERS

- Deploy to the private on-premise infrastructure
- Perform real-time predictions over a video stream from the 4K IP camera
- Process 5 images per second for 121 parking lots
- Support on-demand re-training and re-deployment
- Initial training dataset consisted of 200,000+ images (SoftServe's proprietary)
ARCHITECTURAL DECISION [MODEL DEV]

Training Data Ingestion
- NFS used as a storage layer for training data
- Custom image labeling tool for training data augmentation

Data cleansing and normalization
- Custom image processing pipeline written in Python (split image, lens correction, color correction, contrast and brightness correction etc.)

Feature engineering
- Raw image data used for predictions

Model training and selection
- TensorFlow/Python for model training
- Containerized training jobs ran on a VM scheduled by Ansible

Model persistence
- The result models stored in a private GIT repository (MS TFS)
- Ansible used to deploy a model as a dockerized microservice
ARCHITECTURAL DECISION [MODEL SERVING]

New Data Ingestion
• Polling job transfers new images from the edge device

Data validation and Feature extraction
• Same Python transformations re-used in a Docker-based microservice

Model prediction
• Dockerized microservice deployed to a VM

Serving results
• The results sent to RabbitMQ to serve multiple components
CONCLUSIONS

- In ML systems the behaviour is not specified directly in code but is learned from data.

- As the predictive accuracy of a model may degrade as soon as it is put into production, design decisions must be made to support the initial development and transfer to the serving environment of a model and its continuous refinement.

- These design decisions can be identified through concerns associated with steps of the model development and model serving workflows.

- Decisions made in model development affect model serving and vice versa, so data scientists must work together with data engineers, software architects and devops engineers.