CSE 234
Data Systems for Machine Learning

Arun Kumar

Topic 6: ML Deployment and MLOps

Chapter 8.5 of MLSys book
ML Deployment in the Lifecycle

Data acquisition
Data preparation
Feature Engineering
Training & Inference
Model Selection
Serving
Monitoring

ML/AI + Data Systems Infrastructure

Source → Build → Deploy

Data Scientist/ML Engineer
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

❖ Offline ML Deployment
❖ MLOps:
  ❖ Online Prediction Serving
  ❖ The “3 Vs” of MLOps
  ❖ Monitoring and Versioning
❖ Federated ML
Offline ML Deployment

❖ **Given**: A trained prediction function \( f() \); a **set** of (unlabeled) data examples

❖ **Goal**: Apply inference with \( f() \) to all examples *efficiently*
  ❖ Key metrics: *Throughput*, cost, latency

❖ Historically, offline was the most common scenario
  ❖ Still is among most enterprises, sciences, healthcare
  ❖ Typically once a quarter / month / week / day
  ❖ Aka model *scoring* in some settings
Offline ML Deployment: Systems

- Not particularly challenging in most applications
- All ML systems support inference by default

In-memory: 
- scikit-learn
- R

Disk-based files: 
- SAS
- DASK
- MADlib
- Apache Spark
- MLlib

Layered on RDBMS/Spark: 
- Azure Machine Learning
- Amazon SageMaker
- DataRobot
- H2O.ai

Cloud-native: 
- \text{dm}lc
- XGBoost
- Microsoft LightGBM

"AutoML" platforms: 
- TensorFlow
- PyTorch

Decision tree-oriented:

Deep learning-oriented:
Offline ML Deployment: Optimizations

Q: What systems-level optimizations are possible here?

- **Parallelism:**
  - Inference is *embarrassingly parallel* across examples
- **Factorized ML (e.g., in Morpheus):**
  - Push ML computations down through joins
  - Pre-computes some FLOPS and reuses across examples

The example shows GLM inference:

$$x_i = [x_i, R; x_i, U; x_i, M]$$

$$w^T x_i = w^T_R x_i, R + w^T_U x_i, U + w^T_M x_i, M$$
Offline ML Deployment: Optimizations

Q: What systems-level optimizations are possible here?

- More general pre-computation / caching / batching:
  - Factorized ML is a specific form of sharing/caching
  - Other forms of “multi-query optimization” possible

Example: Batched inference for separate GLMs:

\[
X_{n \times d}(w_1)_{d \times 1} \quad \xrightarrow{\text{Example}} \quad X[w_1; w_2; w_3]_{d \times 3}
\]

\[
Xw_2 \quad Xw_3
\]

Reduces memory stalls for X; raises hardware efficiency
Outline

❖ Offline ML Deployment
❖ MLOps:
  ❖ Online Prediction Serving
  ❖ The “3 Vs” of MLOps
  ❖ Monitoring and Versioning
❖ Federated ML
Software Development + IT Operations (DevOps) is a long standing subarea of software engineering.

No uniform definition but loosely, the science+eng. of administering software in “production”

- Fuses many historically separate job roles
- Cloud and “Agile” s/w eng. have revolutionized DevOps
Background: DevOps

https://medium.com/swlh/how-to-become-an-devops-engineer-in-2020-80b8740d5a52
Key Parts of DevOps Stack/Practice

- Logging & Monitoring
- Continuous Integration (CI) & Continuous Delivery (CD)
- Version Control
- Building & Testing
- Infrastructure-as-Code (IaC), including Config. & Policy
- Microservices / Containerization & Orchestration

Content Credit: Manasi Vartak, Verta.AI
https://aws.amazon.com/devops/what-is-devops/
The Rise of MLOps

- MLOps = DevOps for ML-infused software
  - Much harder than for deterministic software!
- Things that matter beyond just ML model codes:
  - Training and validation datasets
  - Data cleaning/prep/featurization codes/scripts
  - Hyperparameters, other training configs
  - Post-inference rules/configs/ensembling
  - Software versions/configs?
  - Training hardware/configs?

Content Credit: Manasi Vartak, Verta.AI
The Rise of MLOps

❖ Need to change DevOps for ML program semantics
❖ Online Prediction Serving
❖ Logging & Monitoring:
  ❖ Prediction failures; concept drift; feature inflow changes
❖ Version Control:
  ❖ Anything can change: ML code, data, configs, etc.
❖ Build & Test; CI & CD:
  ❖ Rigorous train-val-test splits; beware insidious overfitting
❖ New space with a lot of R&D; no consensus on standards

Content Credit: Manasi Vartak, Verta.AI
The “3 Vs of MLOps”

Operationalizing Machine Learning: An Interview Study

Shreya Shankar*, Rolando Garcia*, Joseph M. Hellerstein, Aditya G. Parameswaran
University of California, Berkeley

❖ Velocity:
  ❖ Need for rapid experimentation, prototyping, and deployment with minimal friction

❖ Validation:
  ❖ Need for checks on quality and integrity of data, features, models, predictions

❖ Versioning:
  ❖ Need to keep track of deployed models and features to ensure provenance and fallback options

The “3 Vs of MLOps”

Operationalizing Machine Learning: An Interview Study

Shreya Shankar*, Rolando Garcia*, Joseph M. Hellerstein, Aditya G. Parameswaran
University of California, Berkeley

- Interplay/tussles between the 3 Vs shapes decisions on tools, processes, and people management in MLOps

- **Examples:**
  - Should Jupyter notebooks be deployed to production? Velocity vs. Validation
  - Are feature stores needed? Velocity vs. Versioning
  - Relabel/augment val. data? Validation vs. Versioning

Birds-eye View of MLOps

Machine Learning Operations (MLOps): Overview, Definition, and Architecture

Dominik Kreuzberger  
KIT  
Germany

Niklas Kühl  
KIT  
Germany

Sebastian Hirschl  
IBM  
Germany

- **Data Scientist**  
  (ML model development)

- **Backend Engineer**  
  (ML infrastructure management)

- **ML Engineer / MLOps Engineer**  
  (cross-functional management of ML environment and assets: ML infrastructure, ML models, ML workflow pipelines, data ingestion, monitoring)

- **Data Engineer**  
  (data management, data pipeline management)

- **DevOps Engineer**  
  (Software engineer with DevOps skills, ML workflow pipeline orchestration, CI/CD pipeline management, monitoring)

- **Software Engineer**  
  (applies design patterns and coding guidelines)

Birds-eye View of MLOps

PRINCIPLES

P1 CI/CD automation
P2 Workflow orchestration
P3 Reproducibility
P4 Versioning of data, code, model
P5 Collaboration
P6 Continuous ML training & evaluation
P7 ML metadata tracking
P8 Continuous monitoring
P9 Feedback loops

COMPONENT

Birds-eye View of MLOps

(See the paper PDF for full res)
Outline

❖ Offline ML Deployment
❖ MLOps:
  ❖ Online Prediction Serving
    ❖ Monitoring and Versioning
❖ Federated ML
Online Prediction Serving

- Standard setting for Web and IoT deployments of ML
  - Usually need to be *realtime*; < 100s of milliseconds!
  - Aka *model serving*

- **Given:** A trained prediction function f(); a *stream* of unlabeled data example(s)

- **Goal:** Apply f() to all/each example *efficiently*
  - Key metrics: *Latency*, *memory footprint*, cost, throughput
Online Prediction Serving

- Surprisingly challenging to do well in ML systems practice!
  - Still an immature area; lot of R&D; many startups

- Key Challenges:
  - **Heterogeneity** of environments: webpages, cloud-based apps, mobile apps, vehicles, IoT, etc.
  - **Unpredictability** of load: need to elastically upscale or downscale resources
  - **Function’s complexity**: model, featurization and data prep code, output thresholds, etc.
    - May straddle libraries, dependencies, even PLs!
    - Hard to optimize end to end in general
The Rise of Serverless Infra.

- Prediction serving is now a “killer app” for Function-as-a-Service (FaaS), AKA serverless cloud infra.
  - Extreme pay-as-you-go; can rent at millisecond level!

- Still, many open efficiency issues for ML deployment:
  - Memory footprints, input access restrictions, logging / output persistence restrictions, latency
Online Prediction Serving: Systems

- A variety of ML serving systems have sprung up recently

**General-purpose** (supports multiple ML tools):

- Amazon SageMaker
- Azure Machine Learning
- Clipper
- Verta

**ML system-specific:**

- TensorFlow Extended
- TF Serving
- PyTorch
- TorchServe
Clipper

- A pioneering general-purpose ML serving system
Generality and modularity:
- One of the first to use containers for ML serving
- Supports multiple ML tools in unified layered API

Efficiency:
- Some basic optimizations: batching to raise throughput; caching of frequently access models/vectors

Multi-model deployment and flexibility:
- A heuristic “model selection” layer to dynamically pick among multiple deployed models; ensembling
Your Reviews on Clipper Paper

- (Walked through in class)
TF Serving is a mature ML serving system, also pioneering
- Optimized for TF model formats; also supports batching
- Dynamic reloading of weights; multiple data sources

TF Lite and TF.JS optimized for more niche backends/runtime environments
Comparing ML Serving Systems

❖ Advantages of general-purpose vs. system-specific:
  ❖ Tool heterogeneity is a reality for many orgs
  ❖ More nimble to customize accuracy post-deployment with different kinds of models/tools
  ❖ Flexibility to swap ML tools; no “lock-in”

❖ Advantages of ML system-specific vs general-purpose:
  ❖ Generality may not be needed (e.g., Google); lower complexity of MLOps
  ❖ Likely more amenable to code/pipeline optimizations
  ❖ Likely better hardware utilization, lower cloud costs
Outline

❖ Offline ML Deployment
❖ MLOps:
  ❖ Online Prediction Serving
  ❖ Monitoring and Versioning
❖ Federated ML
Example for ML Monitoring: TFX

❖ TFX’s “Model Analysis” lets user specify metrics, track over time automatically, alert on-call
❖ Can specify metrics for feature-based data “slices” too

https://www.tensorflow.org/tfx/guide/tfma
Example for ML Monitoring: Overton

- Envisions “code-free” ML monitoring for appl. engineers
- Decouples prediction appl. “task schema” and data
- Emphasizes monitoring of critical training subsets, specifiable using “tags” and “slices”

Example for ML Versioning: Verta

- Started with ModelDB for storing and tracking ML artifacts
  - ML code; data; configuration; environment
- APIs as hooks into ML dev code; SDK and web app./GUI
- Registry for versions and workflows

https://blog.verta.ai/blog/the-third-wave-of-operationalization-is-here-mlops
Open Research Questions in MLOps

- Efficient and consistent version control for ML datasets and featurization codes
- Automate ML prediction failure detection and recovery
- Detect concept drift in an actionable manner; prescribe fixes
- Velocity and complexity of streaming ML applications
- Seamless CI & CD for mass-produced models without insidious overfitting
- Automated end-to-end optimizations spanning feature stores and model serving infrastructure
- …
**Example: MLOps Insights from TFX Traces**

**Production Machine Learning Pipelines:**
**Empirical Analysis and Optimization Opportunities**

Doris Xin
University of California, Berkeley

Hui Miao
Google
huimiao@google.com

Aditya Parameswaran
University of California, Berkeley

Neoklis Polyzotis
Google
npolyzotis@google.com

(a) A simple TFX pipeline comprising three operators: **ExampleGen**, **Trainer**, and **Pusher**. Edges denote the input/output dependencies.

(b) A more typical TFX pipeline that comprises additional operators for data preprocessing, data and model validation, and tuning. Shaded operators correspond to the additional functionality compared to (a).

**Figure 1: Examples of TFX pipelines**

(a) A simple trace for the pipeline in Fig. 1(a).

(b) A real-world trace from our corpus, using operators shown in Fig. 1(b).

**Figure 2: Examples of pipeline traces.** The left-to-right order preserves the time of artifact generation.

Example: MLOps Insights from TFX Traces

(a) Distribution of pipeline span.
(b) Distribution of trained models per day.
(c) Distribution of the number of features.
(d) Distribution of pipeline span.
(e) Distribution of trained models per day.
(f) Distribution of the number of features.

Figure 3: Pipeline Activity and Data Complexity Analysis Results.

Example: MLOps Insights from TFX Traces

Figure 5: Percentage of Trainer runs with each model type

Figure 6: Percentage of pipelines with different operators.

Figure 7: Compute cost of different operators.
Outline

❖ Offline ML Deployment
❖ Online Prediction Serving
❖ ML Monitoring and Versioning
❖ Federated ML
Federated ML

- Pioneered by Google for ML/AI applications on smartphones
- Key benefit is more **user privacy**: 
  - User’s (labeled) data does not leave their device 
  - Decentralizes ML model training/finetuning to user data

https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
Federated ML

- **Key challenge:** Decentralize SGD to intermittent updates
- They proposed a simple “federated averaging” algorithm

\[
\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w).
\]

\[
f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).
\]

- User-partitioned updates breaks IID assumption; skews arise
- Turns out SGD is still pretty robust (recall async. PS); open theoretical questions still being studied

Federated ML

- Privacy/security-focused improvements:
  - New SGD variants; integration with differential privacy
  - Cryptography to anonymize update aggregations
  - Apart from strong user privacy, communication and energy efficiency also major concerns on battery-powered devices

- Systems+ML heuristic optimizations:
  - Compression and quantization to save upload bandwidth
  - Communicate only “high quality” model updates
  - Novel federation-aware ML algorithmics

Federated ML protocol has become quite sophisticated to ensure better stability/reliability, accuracy, and manageability.
Google has neatly abstracted the client-side (embedded in mobile app.) and server-side functionality with actor design.
Federated ML

- Notion of “FL Plan” and simulation-based tooling for data scientists to tailor ML for this deployment regime
- (Users’) Training data is out of reach!
- Model is updated asynchronously automatically
- Debugging and versioning became even more difficult

[Diagram showing development and production environments with Model Program, FL Plan, FL Server, and analytics]

1. Briefly explain 2 reasons why online prediction serving is typically more challenging in practice than offline deployment.

2. Briefly describe 2 systems optimizations performed by Clipper for prediction serving.

3. Briefly discuss 1 systems-level optimization amenable to both offline ML deployment and online prediction serving.

4. Name 3 things that must be versioned for rigorous version control in MLOps.

5. Briefly explain 2 reasons why ML monitoring is needed.

6. Briefly explain 2 reasons why federated ML is more challenging for data scientists to reason about.
Peer Instruction Activity

(Switch slides)
Thank you for taking CSE 234!

Please do submit your course evals before 5pm PT Sunday.
Additional Content
(Optional; not included for Final)
Additional Readings/Resources

https://arxiv.org/abs/2108.07258
On the Opportunities and Risks of Foundation Models

https://www.stateof.ai/
State of AI Report 2022
Hummingbird: Classical ML on DL Tools

- An optimizing compiler to convert classical ML inference computations, especially *tree-based methods*, to tensor ops to exploit DL runtimes, GPU/TPU, etc.
- Branch-heavy instructions -> dense tensor arithmetic
Interestingly, it pays off to embed “useless” calculations in tensor (beyond what is exactly needed for tree) due to massive parallelism of tensor backends!

Slower on 1 or few examples; faster on larger batches

2x-3x faster than SKLearn/ONNX on CPU; 10x on GPU

Figure 3: Compiling an example decision tree using the GEMM strategy (algorithm 1).
Uber’s PyML

https://eng.uber.com/michelangelo-pyml/
Uber’s PyML

- Older approach had coupled models with Java-based online prediction service, reducing flexibility.

Michelangelo

- Trained by Michelangelo via Apache Spark
- Fixed set of supported algorithms
- Supports consistent training-serving preprocessing via fixed DSL
- No dependency isolation

Michelangelo PyML

- Trained by user
- Supports any custom Python model
- Supports custom Python-based preprocessing at serving time
- Full dependency isolation

Replicate high-QPS online models in Michelangelo prior to full-scale rollout

https://eng.uber.com/michelangelo-pyml/