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

- Historically, “offline” was the most common scenario
  - Still is among most enterprises, sciences, healthcare
  - Typically once a quarter / month / week / day
  - Aka “model scoring”

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

- **Goal:** Apply f to all examples *efficiently*
  - Key metrics: *Throughput*, cost, latency
Offline ML Deployment: Systems

- Not particularly challenging in most applications
- Typically all ML systems support it by default

In-memory:
- scikit-learn
- R

Disk-based files:
- SAS
- DASK

Layered on RDBMS/Spark:
- MADlib
- Apache Spark
- MLlib

Cloud-native:
- Azure Machine Learning
- Amazon SageMaker

“AutoML” platforms:
- DataRobot
- H2O.ai

Decision tree-oriented:
- XGBoost
- LightGBM

Deep learning-oriented:
- TensorFlow
- PyTorch
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

\[
x_i = [x_{i,R}; x_{i,U}; x_{i,M}]
\]

**Example:** GLM inference:

\[
\begin{align*}
\mathbf{w}^T \mathbf{x}_i &= \mathbf{w}_R^T \mathbf{x}_{i,R} + \mathbf{w}_U^T \mathbf{x}_{i,U} + \mathbf{w}_M^T \mathbf{x}_{i,M} \\
\end{align*}
\]
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: Bulk inference for separate GLMs:

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

\[
Xw_2 \quad Xw_3
\]

Reduces memory stalls for X; raises hardware efficiency
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!

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

Slower on 1 or few examples; faster on larger batches

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

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

- Very common among Web companies
  - Usually need to be *realtime*; < 100s of milliseconds!
  - Aka “model serving”
- **Given:** A trained ML 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
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” 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
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
Clipper: Principles and Techniques

- **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
Discussion on Clipper paper
Uber’s PyML

https://eng.uber.com/michelangelo-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/
TensorFlow Serving

- 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/runtimes environments
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, serverless costs
Outline

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

- 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

- Monitoring & Logging
- 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 prediction code
  - Much harder than for deterministic software!
- Things that matter beyond just ML model code:
  - Training dataset
  - Data prep/featurization pipelines
  - Hyperparameters
  - Post-inference config. thresholds? Ensembling?
  - Software versions/config.? 
  - Training hardware/config.? 

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

❖ Need to extend DevOps to ML semantics
❖ Monitoring & Logging:
   ❖ Prediction failures? Concept drift? Feature deprecation?
❖ Version Control:
   ❖ Anything can change: ML code + data + config. + … !
❖ Build & Test; CI & CD:
   ❖ Disciplined train-val-test splits? Insidious overfitting?
❖ New space with a lot of R&D; no consensus on standards
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

Example for ML Monitoring: TFX

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.erta.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 pipelines
- Detect concept drift in an actionable manner; prescribe fixes
- Automate ML prediction failure recovery
- Velocity and complexity of streaming ML applications
- Seamless CI & CD for mass-produced models without insidious overfitting
- ...

32
Outline

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

- Pioneered by Google for ML appl. on smartphones
- Key benefit is more **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
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 optimizations:
  - Communicate only “high quality” model updates
  - Compression and quantization to save upload bandwidth
  - New federation-aware ML algorithms

Federated ML protocol has become quite sophisticated to ensure better stability/reliability, accuracy, and manageability.

Figure 1: Federated Learning Protocol

Google has neatly abstracted the client-side (embedded in mobile app.) and server-side functionality with actor design.

Figure 3: Actors in the FL Server Architecture

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 of model program, development environment, production environment, FL Plan, FL Server, analytics, download plan & model, upload model & metrics]

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

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

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

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

Briefly explain 2 reasons why ML monitoring is needed.

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