Architecture of ML Systems
02 Languages, Architectures, and System Landscape

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management
Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - [https://tugraz.webex.com/meet/m.boehm](https://tugraz.webex.com/meet/m.boehm)

- **#2 Course Registrations** (as of Mar 11)
  - Architecture of Machine Learning Systems (AMLS)

- **#3 Study Abroad Fair 2021**
  - Welcome Center: Study Abroad Fair 2021, **Mar 17, 10am**
  - [https://tu4u.tugraz.at/go/study-abroad-fair-2021](https://tu4u.tugraz.at/go/study-abroad-fair-2021)

- **#4 SIGMOD Programming Context 2021**
  - Task: entity resolution pipeline (precision/recall), **Apr 25**
  - [https://dbgroup.ing.unimo.it/sigmod21contest/](https://dbgroup.ing.unimo.it/sigmod21contest/)
Agenda

- Data Science Lifecycle
- ML Systems Stack
- Language Abstractions
- ML Systems Benchmarks
- Programming Projects
Data Science Lifecycle
The Data Science Lifecycle

Data-centric View:
- Application perspective
- Workload perspective
- System perspective

Data Science Lifecycle

- Data Integration
  - Data Cleaning
  - Data Preparation

- Model Selection
  - Training
  - Hyper-parameters

- Validate & Debug
  - Deployment
  - Scoring & Feedback

Exploratory Process
(experimentation, refinements, ML pipelines)

Data Scientist

Data/SW Engineer

DevOps Engineer
The Data Science Lifecycle, cont.

- **Classic KDD Process** (Knowledge Discovery in Databases)
  - Descriptive (association rules, clustering) and predictive

[Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery in Databases. *AI Magazine 17*(3) (1996)]
The Data Science Lifecycle, cont.

- **CRISP-DM**
  - **CRoss-Industry Standard Process for Data Mining**
  - Additional focus on business understanding and deployment

[https://statistik-dresden.de/archives/1128](https://statistik-dresden.de/archives/1128)
The 80% Argument

- **Data Sourcing Effort**
  - Data scientists spend **80-90% time** on finding relevant datasets and data integration/cleaning.

- **Technical Debts in ML Systems**
  - Glue code, pipeline jungles, dead code paths
  - Plain-old-data types, multiple languages, prototypes
  - Abstraction and configuration debts
  - Data testing, reproducibility, process management, and cultural debts


A Text Classification Scenario

- **Example ML Pipeline**
  - Feature Extraction
    - (e.g., doc structure, sentences, tokenization, n-grams)
  - Sentences
  - Token Features
  - Sentence Classification
    - (e.g., ⨝, ∪)
  - “Model”
    - (weights, meta data)

- **Training and Scoring**
  - **Training**
    - FX, FY
    - transformencode
    - MX, MY, B
  - **Scoring**
    - ΔFX, ΔFY
    - transformapply
    - ΔX, ΔŶ
    - transformdecode
  - large-scale, distributed training
  - embedded scoring
ML Systems Stack
What is an ML System?

ML System

- Classification
- Regression
- Recommenders
- Clustering
- Dim Reduction
- Neural Networks

Rapidly Evolving

ML Applications
(entire KDD/DS lifecycle)

- Statistics
- Data Mining
- Machine Learning (ML)

Runtime Techniques
(Execution, Data Access)

- Distributed Systems
- Data Management
- Operating Systems

ML Systems Stack

- HPC
- Compilation Techniques
- Prog. Language Compilers

Accelerators
Driving Factors for ML

- **Improved Algorithms and Models**
  - Success across data and application domains (e.g., health care, finance, transport, production)
  - More complex models which leverage large data

- **Availability of Large Data Collections**
  - Increasing automation and monitoring → data (simplified by cloud computing & services)
  - Feedback loops, simulation/data prog./augmentation → Trend: self-supervised learning

- **HW & SW Advancements**
  - Higher performance of hardware and infrastructure (cloud)
  - Open-source large-scale computation frameworks, ML systems, and vendor-provides libraries

[Credit: Andrew Ng’14]
## Stack of ML Systems

<table>
<thead>
<tr>
<th>ML Systems Stack</th>
<th>Training</th>
<th>Validation &amp; Debugging</th>
<th>Deployment &amp; Scoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hyper-parameter Tuning</strong></td>
<td>ML Apps &amp; Algorithms</td>
<td>Supervised, unsupervised, RL</td>
<td>CPUs, NUMA, GPUs, FPGAs, ASICs, RDMA, SSD/NVM</td>
</tr>
<tr>
<td><strong>Model and Feature Selection</strong></td>
<td>Language Abstractions</td>
<td>Eager interpretation, lazy evaluation, prog. compilation</td>
<td></td>
</tr>
<tr>
<td><strong>Data Programming &amp; Augmentation</strong></td>
<td>Fault Tolerance</td>
<td>Approximation, lineage, checkpointing, checksums, ECC</td>
<td></td>
</tr>
<tr>
<td><strong>Data Preparation (e.g., one-hot, binning)</strong></td>
<td>Execution Strategies</td>
<td>Local, distributed, cloud (data, task, parameter server)</td>
<td></td>
</tr>
<tr>
<td><strong>Data Integration &amp; Data Cleaning</strong></td>
<td>Data Representations</td>
<td>Dense &amp; sparse tensor/matrix; compress, partition, cache</td>
<td></td>
</tr>
</tbody>
</table>

- Improve **accuracy** vs. **performance** vs. **resource requirements**
- Specialization & Heterogeneity

- **Eager interpretation, lazy evaluation, prog. compilation**
- **Approximation, lineage, checkpointing, checksums, ECC**
- **Local, distributed, cloud (data, task, parameter server)**
- **Dense & sparse tensor/matrix; compress, partition, cache**
- **CPUs, NUMA, GPUs, FPGAs, ASICs, RDMA, SSD/NVM**
Accelerators (GPUs, FPGAs, ASICs)

- **Memory- vs Compute-intensive**
  - **CPU**: dense/sparse, large mem, high mem-bandwidth, moderate compute
  - **GPU**: dense, small mem, slow PCI, very high mem-bandwidth / compute

- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: “tensor cores” for 4x4 mm → 64 2B FMA instruction

- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft Catapult/Brainwave Neural Processing Units (NPUs)

- **Application-Specific Integrated Circuits (ASIC)**
  - Spectrum of chips: DL accelerators to computer vision
  - Examples: Google TPUs (64K 2B FMA), NVIDIA DLA, Intel NNP, IBM TrueNorth

- **Quantum Computers?**
  - Examples: IBM Q (Qiskit), Google Sycamore (Cirq → TensorFlow Quantum)
Data Representation

- **ML- vs DL-centric Systems**
  - **ML:** dense and sparse matrices or tensors, different sparse formats (CSR, CSC, COO), frames (heterogeneous)
  - **DL:** mostly dense tensors, relies on embeddings for NLP, graphs

- **Data-Parallel Operations for ML**
  - Distributed matrices: RDF<MatrixIndexes,MatrixBlock>
  - Data properties: distributed caching, partitioning, compression

- **Lossy Compression ➔ Acc/Perf-Tradeoff**
  - Sparsification (reduce non-zero values)
  - Quantization (reduce value domain), learned
  - Data types: bfloat16, Intel Flexpoint (mantissa, exp)
Execution Strategies

- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators

- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend 2018)

- **Lots of PS Decisions ➔ Acc/Perf-Tradeoff**
  - Configurations (#workers, batch size/param schedules, update type/freq)
  - Transfer optimizations: lossy compression, sparsification, residual accumulation, gradient clipping, and momentum corrections
Fault Tolerance & Resilience

- **Resilience Problem**
  - Increasing error rates at scale (soft/hard mem/disk/net errors)
  - Robustness for preemption
  - **Need cost-effective resilience**

- **Fault Tolerance in Large-Scale Computation**
  - Block replication (min=1, max=3) in distributed file systems
  - ECC; checksums for blocks, broadcast, shuffle
  - Checkpointing (MapReduce: all task outputs; Spark/DL: on request)
  - Lineage-based recomputation for recovery in Spark

- **ML-specific Schemes** (exploit app characteristics)
  - Estimate contribution from lost partition to avoid strugglers
  - Example: user-defined “compensation” functions
Language Abstractions

- **Optimization Scope**
  - #1 *Eager Interpretation* (debugging, no opt)
  - #2 *Lazy expression evaluation* (some opt, avoid materialization)
  - #3 *Program compilation* (full opt, difficult)

- **Optimization Objective**
  - Most common: \( \text{min time} \) s.t. memory constraints
  - Multi-objective: \( \text{min cost} \) s.t. time, \( \text{min time} \) s.t. acc, \( \text{max acc} \) s.t. time

- **Trend: Fusion and Code Generation**
  - Custom fused operations
  - Examples: SystemML, Weld, Taco, Julia, TF XLA, TVM, TensorRT

Sparsity-Exploiting Operator
ML Applications

- **ML Algorithms** *(cost/benefit – time vs acc)*
  - Unsupervised/supervised; batch/mini-batch; first/second-order ML
  - Mini-batch DL: variety of NN architectures and SGD optimizers

- **Specialized Apps: Video Analytics in NoScope** *(time vs acc)*
  - Difference detectors / specialized models for “short-circuit evaluation”

- **AutoML** *(time vs acc)*
  - Not algorithms but tasks (e.g., `doClassify(X, y) + search space`)
  - Examples: MLBase, Auto-WEKA, TuPAQ, Auto-sklearn, Auto-WEKA 2.0
  - AutoML services at Microsoft Azure, Amazon AWS, Google Cloud

- **Data Programming and Augmentation** *(acc?)*
  - Generate *noisy labels for pre-training*
  - Exploit expert rules, simulation models, rotations/shifting, and labeling IDEs (Software 2.0)
Language Abstractions and System Architectures
Landscape of ML Systems

TUPAQ  Mlbase  Tupleware
Emma  Kasen  Cümülön(-D)
Glade  Cumulon  Photon ML
TUPAQ  Mlbase  Tupleware
Emma  Kasen  Cümülön(-D)
Glade  Cumulon  Photon ML

SystemDS

SystemML

Mahout Spark ML MADlib
Orion  Santoku  LibFM
Samsara  SimSQL  BUDS

Scikit-learn  Sherlock  ModelHub
Spark R  Spark ML  MADlib

scikit-learn  Sherlock  ModelHub
Spark R  Spark ML  MADlib

SPSS  SAS  Spark R
Mahout  VW  Spark ML

SPSS  SAS  Spark R
Mahout  VW  Spark ML

TensorDB DeepDive

TensorDB DeepDive

LINVIEW  Hemingway  Velox
Longview  TensorDB

LINVIEW  Hemingway  Velox
Longview  TensorDB

TUPAQ  Mlbase  Tupleware
Emma  Kasen  Cümülön(-D)
Glade  Cumulon  Photon ML

TUPAQ  Mlbase  Tupleware
Emma  Kasen  Cümülön(-D)
Glade  Cumulon  Photon ML

R  Matlab  Julia  Weka
SPSS  SAS  Spark R
Mahout  VW  Spark ML

R  Matlab  Julia  Weka
SPSS  SAS  Spark R
Mahout  VW  Spark ML

Weka  scikit-learn  Sherlock  ModelHub
Spark R  Spark ML  MADlib

Weka  scikit-learn  Sherlock  ModelHub
Spark R  Spark ML  MADlib

Hemingway  Glade  Columbus
F  Brainwash  DeepDive

Hemingway  Glade  Columbus
F  Brainwash  DeepDive

OptiML  SystemML  SystemDS

OptiML  SystemML  SystemDS

Bismarck  RAPIDS  Theano
Caffe  DL4J

Bismarck  RAPIDS  Theano
Caffe  DL4J
Language Abstractions and System Architectures

Landscape of ML Systems, cont.

#1 Language Abstraction
- Linear Algebra Programs
- Computation Graphs
- Algorithm Libraries
- Operator Libraries

#2 Execution Strategies
- Parameter Server (Modell-Parallel)
- Task-Parallel Constructs
- Data-Parallel Operations

#3 Distribution
- Local (single node)
- Distributed
- HW accelerators (GPUs, FPGAs, ASICs)

#4 Data Types
- Collections
- Graphs
- Matrices
- Tensors
- Frames
UDF-based Systems

- **User-defined Functions (UDF)**
  - Data type: Input usually collections of cells, rows, or blocks
  - Implement loss and overall optimizer by yourself / UDF abstractions
  - Examples: **data-parallel** (e.g., Spark MLlib) or **In-DBMS analytics** (MADlib, AIDA)

- **Example SQL**
  - **Matrix Product in SQL**
    ```sql
    SELECT A.i, B.j, SUM(A.val*B.val) FROM A, B WHERE A.j = B.i GROUP BY A.i, B.j;
    ```
  - **Matrix Product w/ UDF**
    ```sql
    SELECT A.i, B.j, dot(A.row, B.col) FROM A, B;
    ```
  - **Optimization w/ UDA**
    ```sql
    Init(state) Accumulate(state,data) Merge(state,data) Finalize(state,data)
    ```
Graph-based Systems

- **Google Pregel**
  - Name: Seven Bridges of Koenigsberg (Euler 1736)
  - "Think-like-a-vertex" (vertex-centric processing)
  - Iterative processing in super steps, comm.: message passing

- **Programming Model**
  - Represent graph as collection of vertices w/ edge (adjacency) lists
  - Implement algorithms via Vertex API
  - Terminate if all vertices halted / no more msgs

```java
public abstract class Vertex {
    public String getID();
    public long superstep();
    public VertexValue getValue();
    public compute(Iterator<Message> msgs);
    public sendMessageTo(String v, Message msg);
    public void voteToHalt();
}
```
Graph-based Systems, cont.

- **Example 1: Connected Components**
  - Determine connected components of a graph (subgraphs of connected nodes)
  - Propagate max(current, msgs) if != current to neighbors, terminate if no msgs

- **Example 2: Page Rank**
  - Ranking of webpages by importance/impact
  - #1: Initialize vertices to 1/numVertices()
  - #2: In each super step
    - Compute current vertex value:
      \[
      \text{value} = \frac{0.15}{\text{numVertices}()} + 0.85 \times \text{sum(msg)}
      \]
    - Send to all neighbors:
      \[
      \text{value/numOutgoingEdges()}
      \]

[Credit: https://en.wikipedia.org/wiki/PageRank]
Graph-based Systems, cont.

- **Excursus: Graph Processing via Sparse Linear Algebra**
  
  - SystemDS’ components()
    
    ```r
    # initialize state with vertex ids
c = seq(1,nrow(G));
diff = Inf;
iter = 1;

    # iterative computation of connected components
    while( diff > 0 & (maxi==0 | iter<=maxi) ) {
        u = max(rowMaxs(G * t(c)), c);
diff = sum(u != c);
c = u; # update assignment
    iter = iter + 1;
    }
    
    alpha = ifdef(argAlpha, 0.85);
    while( i < maxi ) {
        # power iteration on G w/ Gij = 1/degree
        p = alpha*(G %% p) + (1-alpha)*(e %% u %% p);
i += 1;
    }
    ```

  - SystemDS’ pageRank()

Linear Algebra Systems

- **Comparison Query Optimization**
  - Rule- and cost-based rewrites and operator ordering
  - Physical operator selection and query compilation
  - Linear algebra / other ML operators, DAGs, control flow, sparse/dense formats

- **#1 Interpretation** (operation at-a-time)
  - Examples: R, PyTorch, Morpheus [PVLDB’17]

- **#2 Lazy Expression Compilation** (DAG at-a-time)
  - Examples: RIOT [CIDR’09], TensorFlow [OSDI’16], Mahout Samsara [MLSystems’16]
  - Examples w/ control structures: Weld [CIDR’17], OptiML [ICML’11], Emma [SIGMOD’15]

- **#3 Program Compilation** (entire program)
  - Examples: SystemML [PVLDB’16], Julia, Cumulon [SIGMOD’13], Tupleware [PVLDB’15]

---

**Optimization Scope**

```r
1: X = read($1); # n x m matrix
2: y = read($2); # n x 1 vector
3: maxi = 50; lambda = 0.001;
4: intercept = $3;
5: ... 
6: r = -(t(X) %*% y);
7: norm_r2 = sum(r * r); p = -r;
8: w = matrix(0, ncol(X), 1); i = 0;
9: while(i<maxi & norm_r2>norm_r2_trgt) 
10: { 
11: q = (t(X) *%* X %*% p)+lambda*p;
12: alpha = norm_r2 / sum(p * q);
13: w = w + alpha * p;
14: old_norm_r2 = norm_r2;
15: r = r + alpha * q;
16: norm_r2 = sum(r * r);
17: beta = norm_r2 / old_norm_r2;
18: p = -r + beta * p; i = i + 1;
19: }
20: write(w, $4, format="text");
```
Language Abstractions and System Architectures

Linear Algebra Systems, cont.

- **Some Examples ...**

Apache SystemML™

\[
X = \text{read}(\text{"./X"});
y = \text{read}(\text{"./y"});
p = t(X) \%\% y;
w = \text{matrix}(0, \text{ncol}(X), 1);
\]

\[
\text{while}(...) \{
q = t(X) \%\% X \%\% p;
... 
\}
\]

(Custom DSL w/ R-like syntax; program compilation)

TensorFlow

\[
\text{X = X.par(256).checkpoint()}
\]

(Embedded DSL in Scala; lazy evaluation)

\[
\begin{align*}
\text{var X = drmFromHDFS(\text{"./X"})} \\
\text{val y = drmFromHDFS(\text{"./y"})} \\
\text{var p = (X.t \%\% y).collect} \\
\text{var w = dense(...)}
\end{align*}
\]

(Embedded DSL in Python; lazy [and eager] evaluation)

\[
\text{while}(...) \{
q = (X.t \%\% X \%\% p).collect \\
... 
\}
\]

(Note: TF 2.0

[Dan Moldovan et al.: AutoGraph: Imperative-style Coding with Graph-based Performance. *SysML 2019.*]
ML Libraries

- **Fixed algorithm implementations**
  - Often on top of existing linear algebra or UDF abstractions

Single-node Example (Python)

```python
from numpy import genfromtxt
from sklearn.linear_model import LinearRegression

X = genfromtxt('X.csv')
y = genfromtxt('y.csv')

reg = LinearRegression()
.reg.fit(X, y)
out = reg.score(X, y)
```

Distributed Example (Spark Scala)

```scala
import org.apache.spark.ml.regression.LinearRegression

val X = sc.read.csv('X.csv')
val y = sc.read.csv('y.csv')
val Xy = prepare(X, y).cache()

val reg = new LinearRegression()
.reg.fit(Xy)
val out = reg.transform(Xy)
```
DNN Frameworks

- **High-level DNN Frameworks**
  - Language abstraction for DNN construction and model fitting
  - Examples: Caffe, Keras

```python
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
... 
```

- **Low-level DNN Frameworks**
  - Examples: TensorFlow, MXNet, PyTorch, CNTK

```python
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test),
          shuffle=True)
```
Feature-centric Tools

- **DeepDive**
  - Knowledge base construction via SQL/MLNs
  - Grounding: SQL queries $\rightarrow$ factor graph
  - Inference: statistical inference on factor graph
  - Incremental maintenance via sampling / variational approach

- **Overton (Apple)**
  - Building, monitoring, improving ML pipelines
  - High-level abstractions: tasks and payloads
  - Data slicing, multi-task learning, data augmentation

- **Ludwig (Uber AI)**
  - Data types and configuration files
  - Encoders, combiners, decoders
  - Example “visual question answering”:


[Christopher Ré et al: Overton: A Data System for Monitoring and Improving Machine-Learned Products, *CIDR 2020*]

ML Systems Benchmarks
“Big Data” Benchmarks w/ ML Components

- **BigBench**
  - 30 workloads (6 statistics, 17 data mining)
  - Different data sources, processing types
  - **Note:** TPCx-BB, TPCx-HS [TPCTC 2016]

- **HiBench (Intel)**
  - MapReduce Micro benchmarks (WC, TeraSort)
  - IR/ML (e.g., PageRank, K-means, Naïve Bayes)

- **GenBase**
  - Preprocessing and ML in array databases

- **SparkBench**
  - Existing library algorithms (ML, Graph, SQL, stream)
  - ML: LogReg, SVM, matrix factorization, PageRank

[Ahmad Ghazal et al: **BigBench**: towards an industry standard benchmark for big data analytics. **SIGMOD 2013**]

[Lan Yi, Jinquan Dai: Experience from Hadoop Benchmarking with **HiBench**: From Micro-Benchmarks Toward End-to-End Pipelines. **WBDB 2013**]

[Rebecca Taft et al: **GenBase**: a complex analytics genomics benchmark. **SIGMOD 2014**]

Linear Algebra and DNN Benchmarks

- **SLAB: Scalable LA Benchmark (UCSD)**
  - **Ops:** TRANS, NORM, GRM, MVM, ADD, GMM
  - **Pipelines/Decompositions:** MMC, SVD
  - **Algorithms:** OLS, LogReg, NMF, HRSE

- **DAWNBench (Stanford)**
  - Image Classification ImageNet: 93% top-5 val err
  - Image Classification CIFAR10: 94% test accuracy
  - Question Answering SQuAD: 0.75 F1 measure

- **MLPerf**
  - Image classification ImageNet, object detection COCO, translation WMT En-Ger, recommendation MovieLens, reinforcement learning GO
  - **Train to target accuracy**


[Peter Mattson et al.: MLPerf Training Benchmark, *MLSys 2020*]
DNN Benchmarks, cont.

**Closed Division Times**

<table>
<thead>
<tr>
<th>#</th>
<th>Submitter</th>
<th>System</th>
<th>Processor #</th>
<th>Accelerator #</th>
<th>Software</th>
<th>Benchmark results (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6.1</td>
<td>Google</td>
<td>TPUv3.32</td>
<td>TPUv3</td>
<td>16</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>42.19, 12.61, 107.03, 12.25, 10.20</td>
</tr>
<tr>
<td>0.6.2</td>
<td>Google</td>
<td>TPUv3.128</td>
<td>TPUv3</td>
<td>64</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>11.22, 3.89, 57.40, 4.62, 3.85</td>
</tr>
<tr>
<td>0.6.3</td>
<td>Google</td>
<td>TPUv3.256</td>
<td>TPUv3</td>
<td>128</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>6.86, 2.76, 35.60, 3.53, 2.81</td>
</tr>
<tr>
<td>0.6.4</td>
<td>Google</td>
<td>TPUv3.512</td>
<td>TPUv3</td>
<td>256</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>3.85, 1.79, 2.51, 1.58</td>
</tr>
<tr>
<td>0.6.5</td>
<td>Google</td>
<td>TPUv3.1024</td>
<td>TPUv3</td>
<td>512</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>2.27, 1.34, 2.11, 1.05</td>
</tr>
<tr>
<td>0.6.6</td>
<td>Google</td>
<td>TPUv3.2048</td>
<td>TPUv3</td>
<td>1024</td>
<td>TensorFlow, TPU 1.14.1.dev</td>
<td>1.28, 1.21, 0.85</td>
</tr>
</tbody>
</table>

**Available on-premise**

| 0.7 | Intel | 32x2S CLX 8260L | CLX 8260L | 64 | TensorFlow | 14.43 | [details] [code] [none] |
| 0.8 | NVIDIA | DGX-1 | Tesla V100 | 8 | MXNet, NC19.05 | 115.22 | [details] [code] [none] |
| 0.9 | NVIDIA | DGX-1 | Tesla V100 | 6 | PyTorch, NC19.05 | 22.36, 207.48, 20.55, 20.34 | [details] [code] [none] |
| 0.10 | NVIDIA | DGX-1 | Tesla V100 | 8 | TensorFlow, NC19.05 | 27.39 | [details] [code] [none] |
| 0.11 | NVIDIA | DGX-1 | Tesla V100 | 24 | TensorFlow, NC19.05 | 13.57 | [details] [code] [none] |
| 0.12 | NVIDIA | 24x1 DGX-1 | Tesla V100 | 120 | PyTorch, NC19.05 | 22.03 | [details] [code] [none] |
| 0.13 | NVIDIA | 30x DGX-1 | Tesla V100 | 240 | PyTorch, NC19.05 | 2.87 | [details] [code] [none] |
| 0.14 | NVIDIA | 48x DGX-1 | Tesla V100 | 884 | PyTorch, NC19.05 | 1.99 | [details] [code] [none] |
| 0.15 | NVIDIA | 60x DGX-1 | Tesla V100 | 480 | PyTorch, NC19.05 | 2.05 | [details] [code] [none] |
| 0.16 | NVIDIA | 130x DGX-1 | Tesla V100 | 1040 | MXNet, NC19.05 | 1.69 | [details] [code] [none] |
| 0.17 | NVIDIA | DGX-2 | Tesla V100 | 16 | MXNet, NC19.05 | 57.87 | [details] [code] [none] |
| 0.18 | NVIDIA | DGX-2 | Tesla V100 | 16 | PyTorch, NC19.05 | 12.21, 101.00, 10.94, 11.04 | [details] [code] [none] |
| 0.19 | NVIDIA | DGX-2H | Tesla V100 | 16 | TensorFlow, NC19.05 | 52.74, 95.20, 9.87, 9.80 | [details] [code] [none] |
| 0.20 | NVIDIA | DGX-2H | Tesla V100 | 16 | PyTorch, NC19.05 | 11.41, 95.20, 9.87, 9.80 | [details] [code] [none] |
| 0.21 | NVIDIA | 4x DGX-2H | Tesla V100 | 64 | PyTorch, NC19.05 | 4.78, 32.72, 2.41 | [details] [code] [none] |
| 0.22 | NVIDIA | 10x DGX-2H | Tesla V100 | 160 | PyTorch, NC19.05 | 2.41 | [details] [code] [none] |
| 0.23 | NVIDIA | 12x DGX-2H | Tesla V100 | 192 | PyTorch, NC19.05 | 18.47 | [details] [code] [none] |
| 0.24 | NVIDIA | 15x DGX-2H | Tesla V100 | 240 | PyTorch, NC19.05 | 2.50 | [details] [code] [none] |
| 0.25 | NVIDIA | 16x DGX-2H | Tesla V100 | 256 | PyTorch, NC19.05 | 2.12 | [details] [code] [none] |
| 0.26 | NVIDIA | 24x DGX-2H | Tesla V100 | 304 | PyTorch, NC19.05 | 1.60 | [details] [code] [none] |
| 0.27 | NVIDIA | 30x DGX-2H, 8 chips each | Tesla V100 | 240 | PyTorch, NC19.05 | 2.23 | [details] [code] [none] |
| 0.28 | NVIDIA | 30x DGX-2H | Tesla V100 | 480 | PyTorch, NC19.05 | 1.59 | [details] [code] [none] |
| 0.29 | NVIDIA | 32x DGX-2H | Tesla V100 | 512 | MXNet, NC19.05 | 2.59 | [details] [code] [none] |
| 0.30 | NVIDIA | 96x DGX-2H | Tesla V100 | 1536 | MXNet, NC19.05 | 1.33 | [details] [code] [none] |

**MLPerf v0.6**: [https://mlperf.org/training-results-0-6/](https://mlperf.org/training-results-0-6/)

**MLPerf v0.7**: [https://mlperf.org/training-results-0-7](https://mlperf.org/training-results-0-7)

96 x DGX-2H = 96 * 16 = 1536 V100 GPUs

⇒ 96 * $400K = $35M – $40M

AutoML and Data Cleaning

- **MLBench**
  - Compare AutoML w/ human experts (Kaggle)
  - Classification, regression; AUC vs Runtime

- **(Open Source) AutoML Benchmark**
  - 39 classification datasets, AUC metric, 10-fold CV
  - Extensible metrics, OS AutoML frameworks, datasets

- **CleanML**
  - Train/Test on dirty vs clean data (2x2)
  - Missing values, outliers, duplicates, mislabels

- **Meta Worlds Benchmark**
  - Meta-reinforcement and multi-task learning
  - 50 robotic manipulation tasks (e.g., get coffee, open window, pick & place)
Programming Projects

Refinement until **March 26**
(bring you own if you want)

**Project Selection by April 02**
Overview Project Types

- **#1 Apache SystemDS Projects**
  - [https://issues.apache.org/jira/secure/Dashboard.jspa?selectPagId=12335852#Filter-Results/12365413](https://issues.apache.org/jira/secure/Dashboard.jspa?selectPagId=12335852#Filter-Results/12365413)
  - Features across the stack (built-in scripts, APIs, compiler, runtime)

- **#2 DAPHNE Projects**
  - Private list of projects, descriptions on demand, OSS ~01/2022
  - Features at level of runtime, compiler, tools

- **#3 Data Cleaning Benchmark**
  - Design and implement new data cleaning benchmark
  - Docs, toolkit (e.g., datagen), and benchmark driver

- **#4 Alternative Exercise: Siemens Student Challenge**
  - ML model for classification w/ dependability assessment
  - (Submission deadline: **May 02**, total prices: **10,000 EUR**)

Apache SystemDS Projects

- **#S1 New built-in functions** (algorithms, NN archs, FNN, GAN, cleaning)
- **#S2 Python API extensions** (frame support, multi-return)
- **#S3 Documentation and Tutorials** (for different target users)
- **#S4 Benchmarks and Tests** (SLAB benchmark, perf/test frameworks)
- **#S5 Lineage-based debugging** (convergence, model behavior, fairness)
- **#S6 Auto Differentiation** (built-in function and compiler)
- **#S7 Loop Vectorization Rewrites** (more general framework)
- **#S8 Extended CSE & Constant Folding** (commutativity, one-shot)
- **#S9 Extended Update In-Place Framework** (reference counting)
- **#S10 Extended Matrix Multiplication Chain Opt** (sparsity, rewrites)
- **#S11 Operator Scheduling Algorithms** (baselines, lazy, async)
- **#S12 Compressed Linear Algebra** (read, constant/delta, functional)
- **#S13 Extended Intel MKL-DNN Runtime Operations** (beyond conv2d)
- **#S14 Extended I/O Framework for Other Formats** (NetCDF, HDF5, Arrow)
DAPHNE Projects

- **#D1** Parser for SystemDS DSL → DaphneIR
- **#D2** Parser for subset of SQL → DaphneIR
- **#D3** Explain: readable IR via custom IR-level parser/printers
- **#D4** Sparsity-aware MM chain optimization w/ rewrites
- **#D5** Various LA and RA simplification rewrites
- **#D6** IO readers/writers for common data formats (arrow, parquet)
- **#D7** Matrix and frame data generators (dense and sparse, properties)
- **#D8** Kernels for LA and RA operations (dense and sparse)
- **#D9** Distributed runtime operations on Spark
- **#D10** Analyze: Extraction of data characteristics (interesting properties)
Summary and Q&A

- Data Science Lifecycle
- ML Systems Stack
- Language Abstractions
- ML System Benchmarks
- Programming Projects (first come, first serve)

- **Recommended Reading** (a critical perspective on a broad sense of ML systems)
  - [M. Jordan: SysML: Perspectives and Challenges. Keynote at SysML 2018]
  - “ML [...] is far from being a solid engineering discipline that can yield robust, scalable solutions to modern data-analytic problems”
  - [https://www.youtube.com/watch?v=4inIBmY8dQI](https://www.youtube.com/watch?v=4inIBmY8dQI)