

Architecture of ML Systems (AMLS)

02 Languages, Architectures, and System Landscape

Prof. Dr. Matthias Boehm

Technische Universität Berlin

Berlin Institute for the Foundations of Learning and Data

Big Data Engineering (DAMS Lab)

Announcements / Org



■ #1 Hybrid & Video Recording

- Hybrid lectures (in-person, zoom) with optional attendance

<https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09>

- Zoom [video recordings](#), links from website

https://mboehm7.github.io/teaching/ss25_aml/index.htm



■ #2 Course Registrations

- TU Berlin: **308** (TUB ISIS registrations)

~310

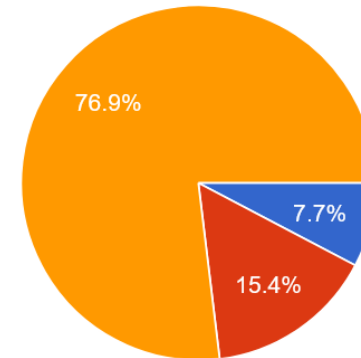
■ #3 Projects / Alternative Exercise

- [Task description](#) on course website since Apr 12

- **Project Selection by May 01**, Submission by **July 08**

■ #4 Upcoming Lectures

- May 02 and May 09, **Fri 4pm-6pm** in EB 407



● SystemDS Project
● DAPHNE Project
● Alternative Exercise

<https://tinyurl.com/yeyrjyxw>



Agenda

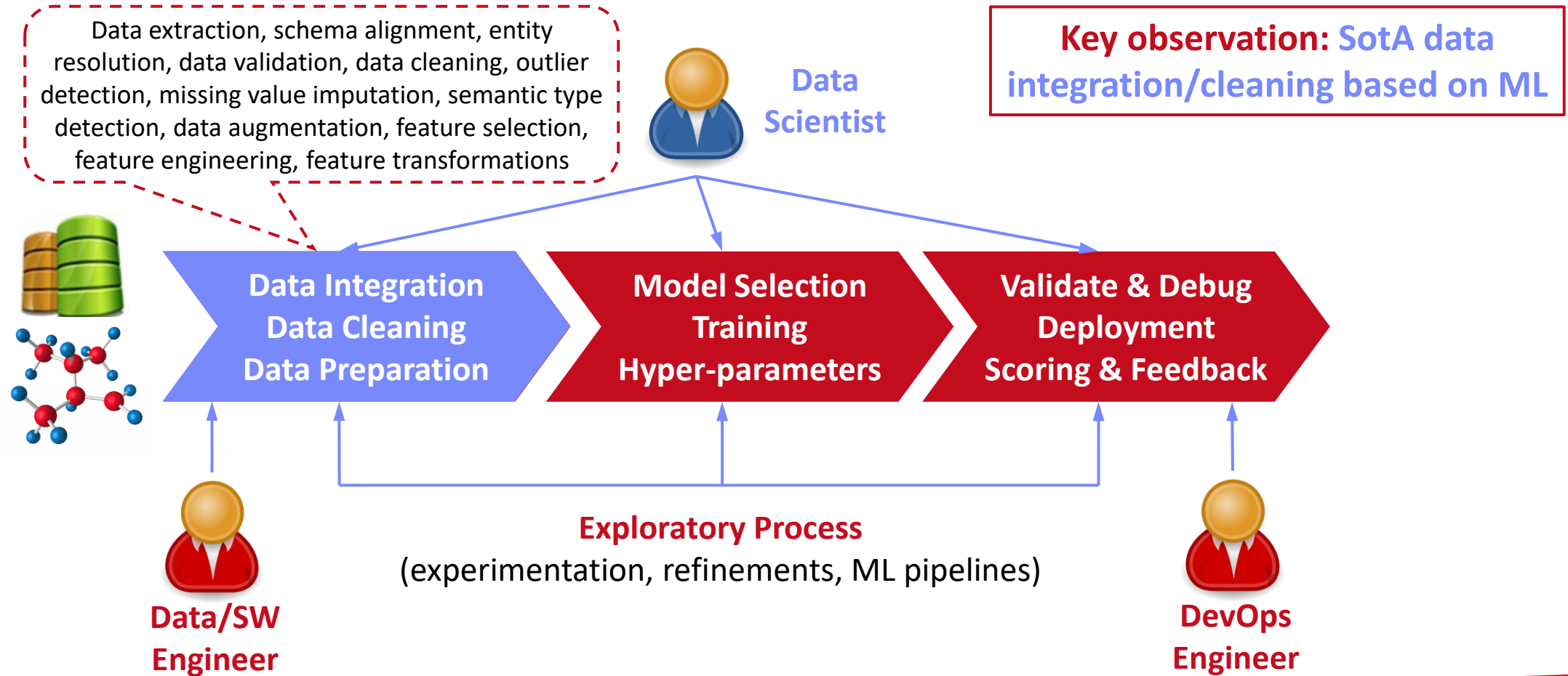


- **Data Science Lifecycle**
- **ML Systems Stack**
- **Language Abstractions**
- **ML Systems Benchmarks**

Data Science Lifecycle

The Data Science Lifecycle (aka KDD Process, aka CRISP-DM)

Data-centric View:
Application/workload/system perspectives

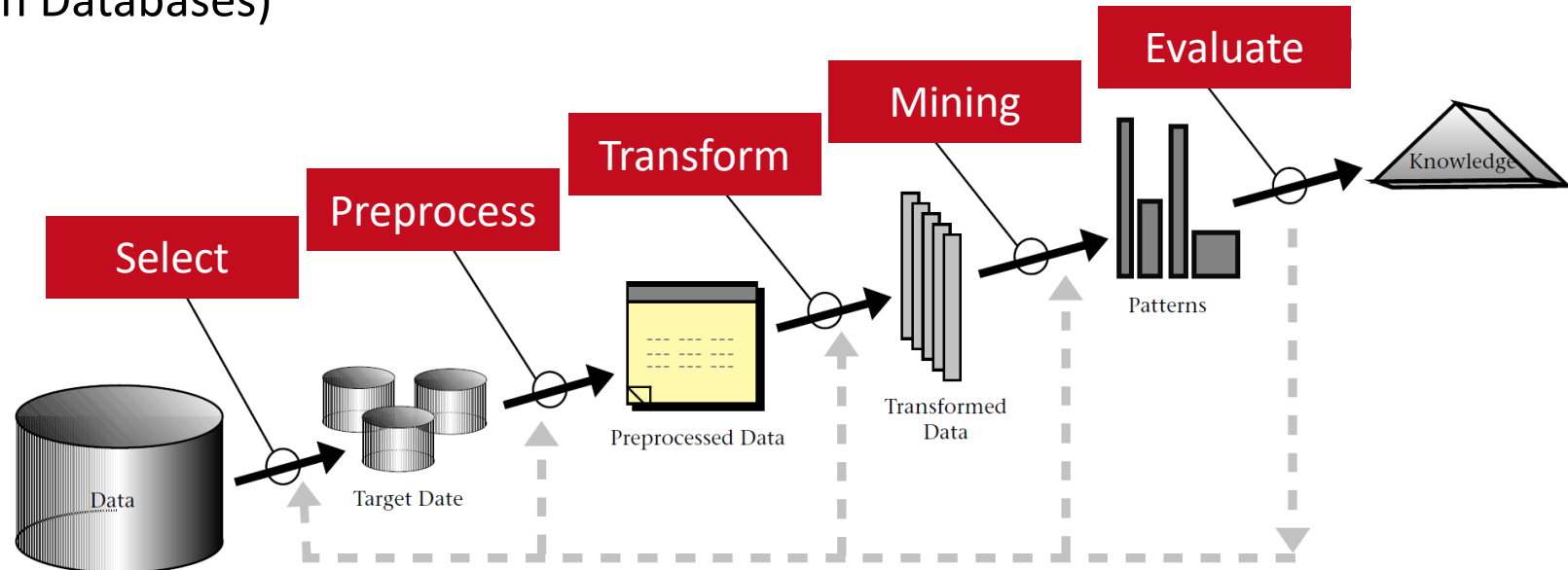


The Data Science Lifecycle, cont.

■ Classic KDD Process

(Knowledge Discovery in Databases)

- Descriptive
(association rules, clustering)
- Predictive



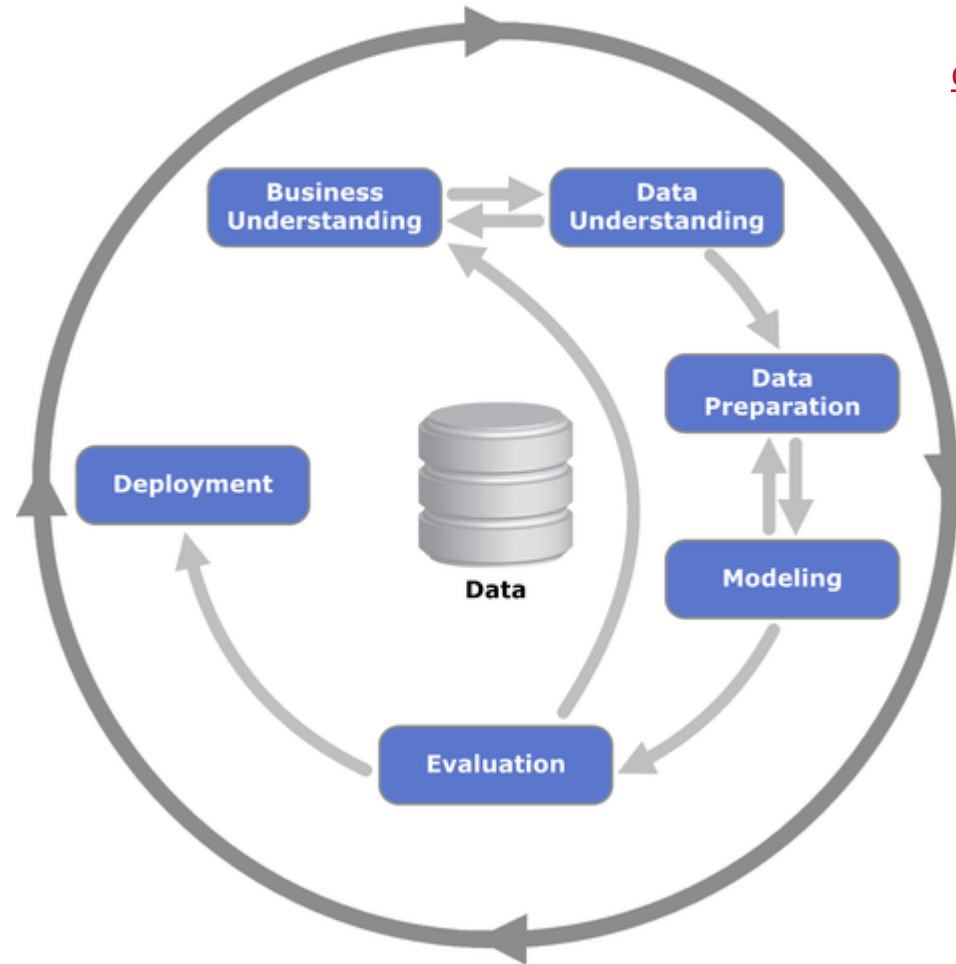
[Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery in Databases. **AI Magazine** 17(3) (1996)]

The Data Science Lifecycle, cont.

■ CRISP-DM

- **C**ross-Industry
Standard **P**rocess for
Data **M**ining
- Additional focus on
business understanding
and deployment

[<https://statistik-dresden.de/archives/1128>]



The 80% Argument



■ Data Sourcing Effort

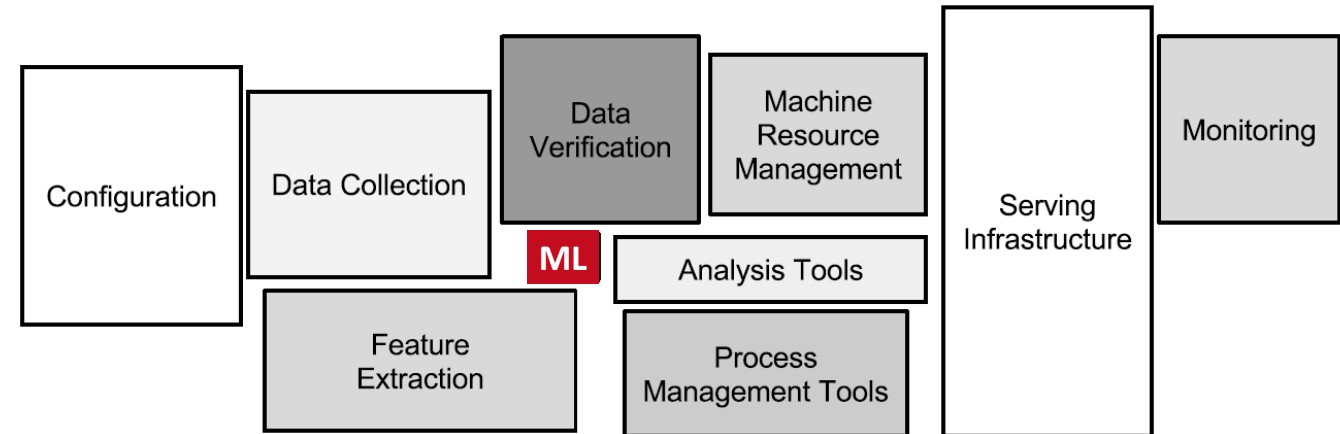
- Data scientists spend **80-90% time** on finding relevant datasets and data integration/cleaning.

[Michael Stonebraker, Ihab F. Ilyas:
Data Integration: The Current
Status and the Way Forward.
IEEE Data Eng. Bull. 41(2) (2018)]



■ 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

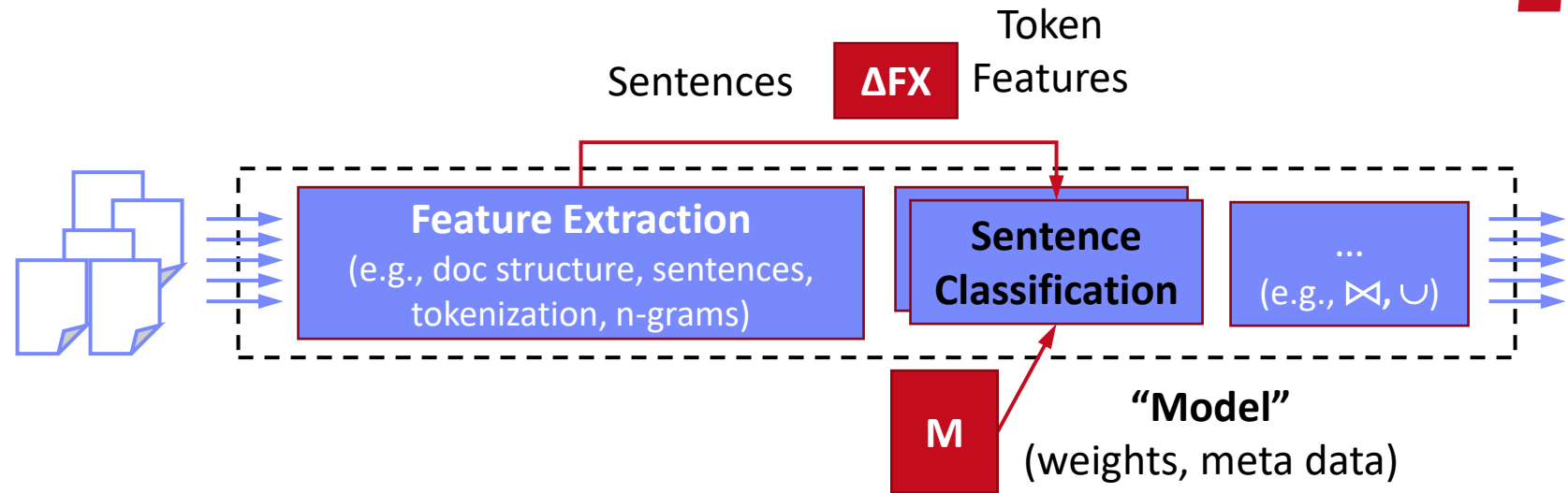


[D. Sculley et al.: Hidden
Technical Debt in Machine
Learning Systems. **NeurIPS 2015**]

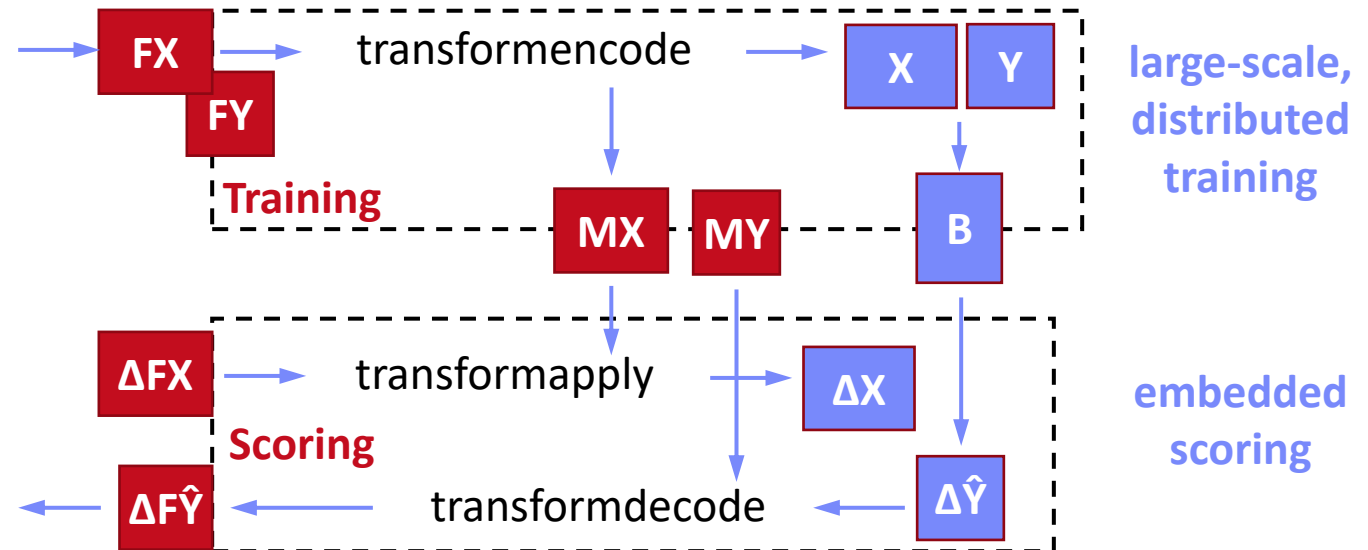


A Text Classification Scenario

■ Example ML Pipeline

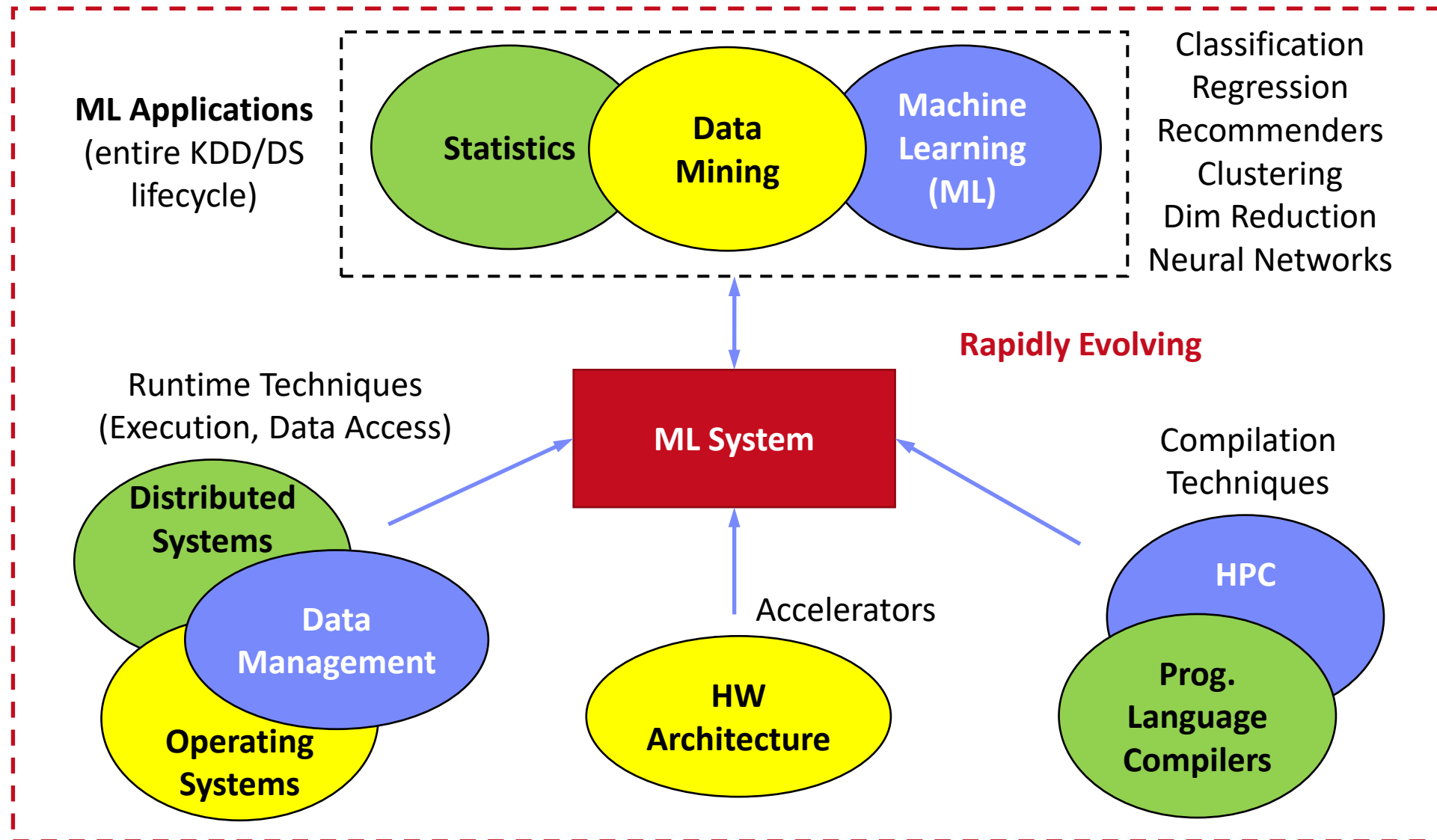


■ Training and Scoring



ML Systems Stack

What is an ML System?



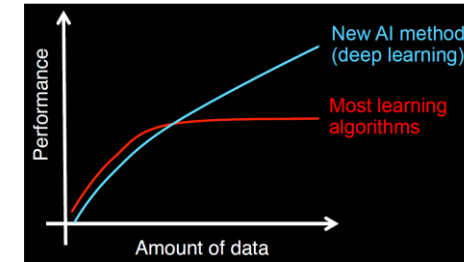
Driving Factors for ML



■ Improved Algorithms and Models

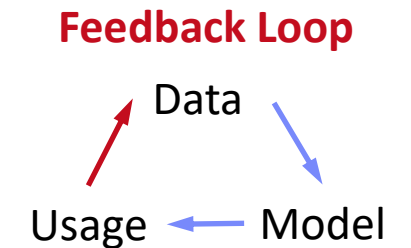
- Success across data and application domains (e.g., health care, finance, transport, production)
- More complex models which leverage large data

[Credit: Andrew Ng'14]



■ Availability of Large Data Collections

- Increasing automation and monitoring → data (simplified by cloud computing & services, annotation services)
- Feedback loops, **simulation/data prog./augmentation**
→ Trend: **self-supervised learning** (*-GPT-x)



■ HW & SW Advancements

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



Stack of ML Systems



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 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

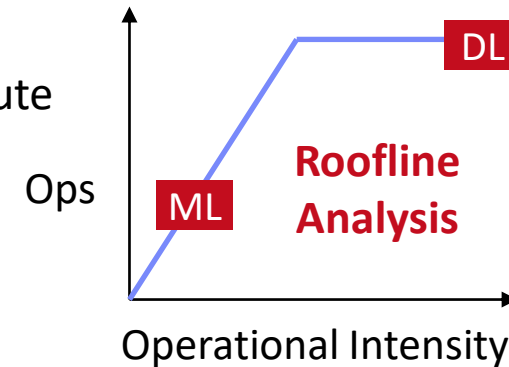
■ Field-Programmable Gate Arrays (FPGAs)

- Customizable HW accelerators for prefiltering, compression, DL
- Examples: Microsoft Catapult/Brainwave Neural Processing Units (NPU)

■ 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: Examples: IBM Q (Qiskit), Google Sycamore (Cirq → TensorFlow Quantum)



Apps
Lang
Faults
Exec
Data
HW

Data Representation



Apps

Lang

Faults

Exec

Data

HW

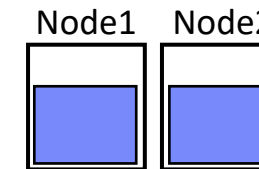
■ ML- vs DL-centric Systems

- **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

Example Word Embedding:
 $\text{vec}(\text{Berlin}) - \text{vec}(\text{Germany}) + \text{vec}(\text{France}) \approx \text{vec}(\text{Paris})$

■ Data-Parallel Operations for ML

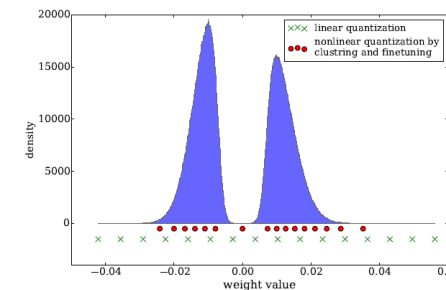
- Distributed matrices: `RDD<MatrixIndexes, MatrixBlock>`
- Data properties: **distributed caching**, **partitioning**, **compression**



■ Lossy Compression → Acc/Perf-Tradeoff

- Sparsification (reduce non-zero values)
- Quantization (reduce value domain), learned
- Data types: **bfloat16**, Intel Flexpoint (mantissa, exp)

[Credit: Song Han'16]



Execution Strategies



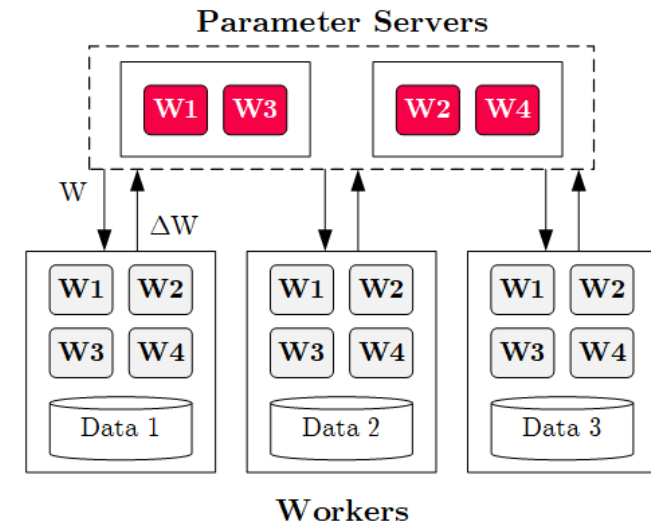
■ Batch Algorithms: Data and Task Parallel

- 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 since 2018)



Apps
Lang
Faults
Exec
Data
HW

■ 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



Apps
Lang
Faults
Exec
Data
HW

■ Resilience Problem

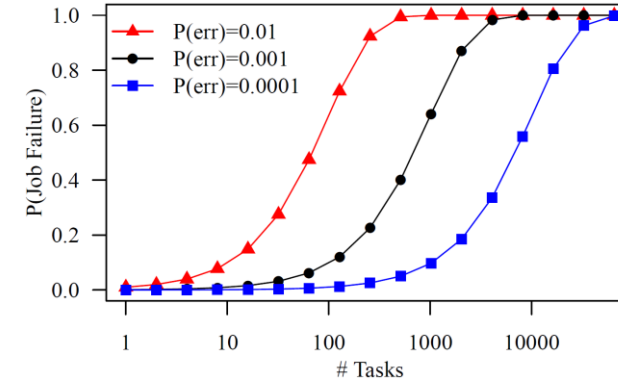
- Increasing error rates at scale (soft/hard mem/disk/net errors)
- Robustness for preemption
- **Need cost-effective resilience**

■ 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 stragglers
- Example: user-defined “compensation” functions



[Bianca Schroeder, Eduardo Pinheiro, Wolf-Dietrich Weber: DRAM errors in the wild: a large-scale field study. **SIGMETRICS 2009**]



[Sebastian Schelter, Stephan Ewen, Kostas Tzoumas, Volker Markl: "All roads lead to Rome": optimistic recovery for distributed iterative data processing. **CIKM 2013**]



Language Abstractions



■ Optimization Scope

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

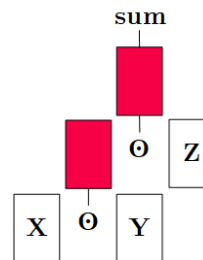


■ Optimization Objective

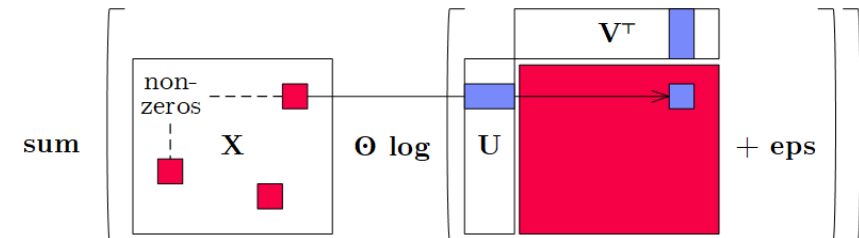
- Most common: **min time** s.t. memory constraints
- Multi-objective: **min cost** s.t. time, **min time** s.t. acc, **max acc** s.t. time

■ Trend: Fusion and Code Generation

- Custom fused operations
- **Examples:** SystemML, Weld, Taco, Julia, TF XLA, TVM, TensorRT



Sparsity-Exploiting Operator



Apps
Lang
Faults
Exec
Data
HW

■ 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

- Difference detectors / specialized models for “short-circuit evaluation”



[Credit: Daniel Kang'17]

■ 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

[Chris Thornton, Frank Hutter, et al:
Auto-WEKA: combined selection and
hyperparameter optimization of
classification algorithms. **KDD 2013**]



■ Data Programming and Augmentation (**acc?**)

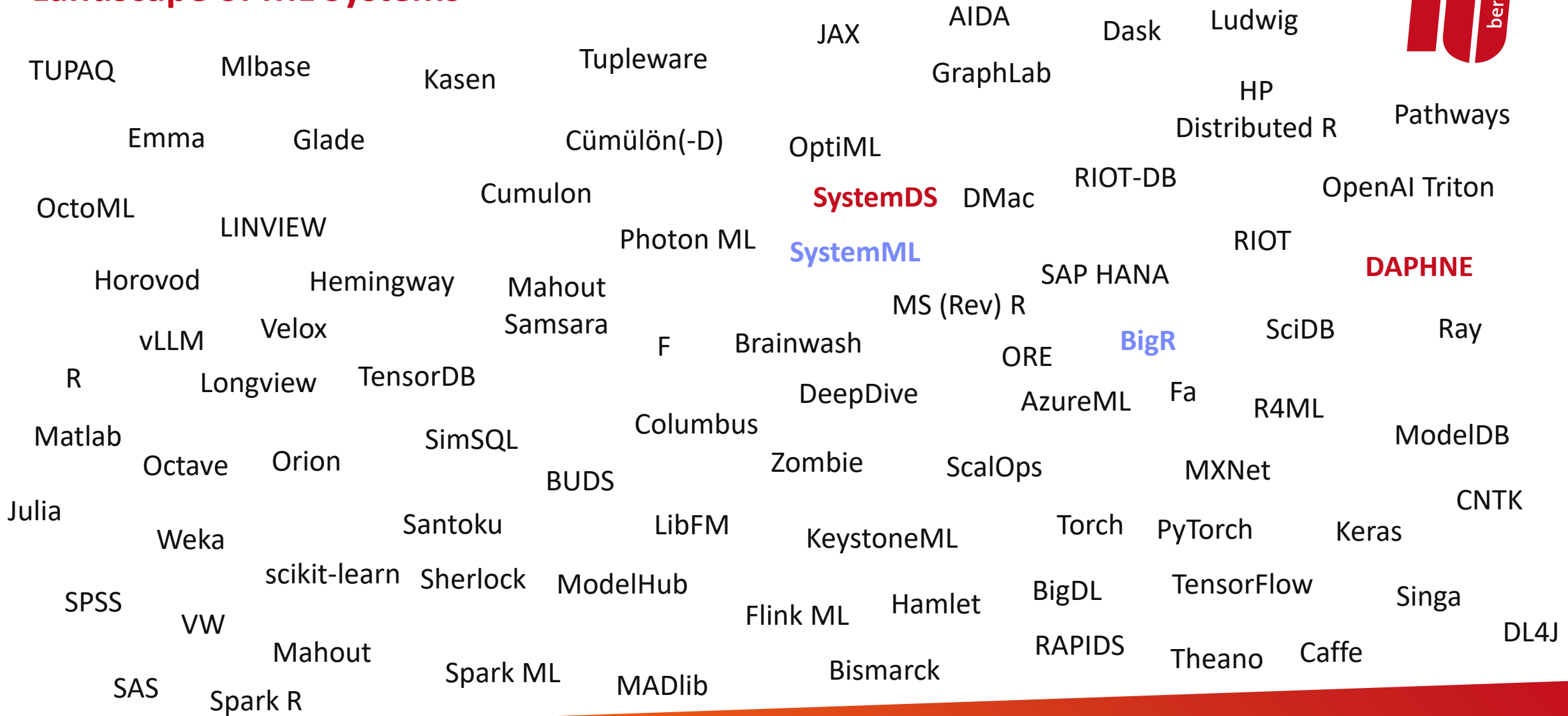
- Generate **noisy labels for pre-training**
- Exploit expert rules, simulation models, rotations/shifting, and labeling IDEs (Software 2.0)

[Credit:
Jonathan
Tremblay'18]



Language Abstractions

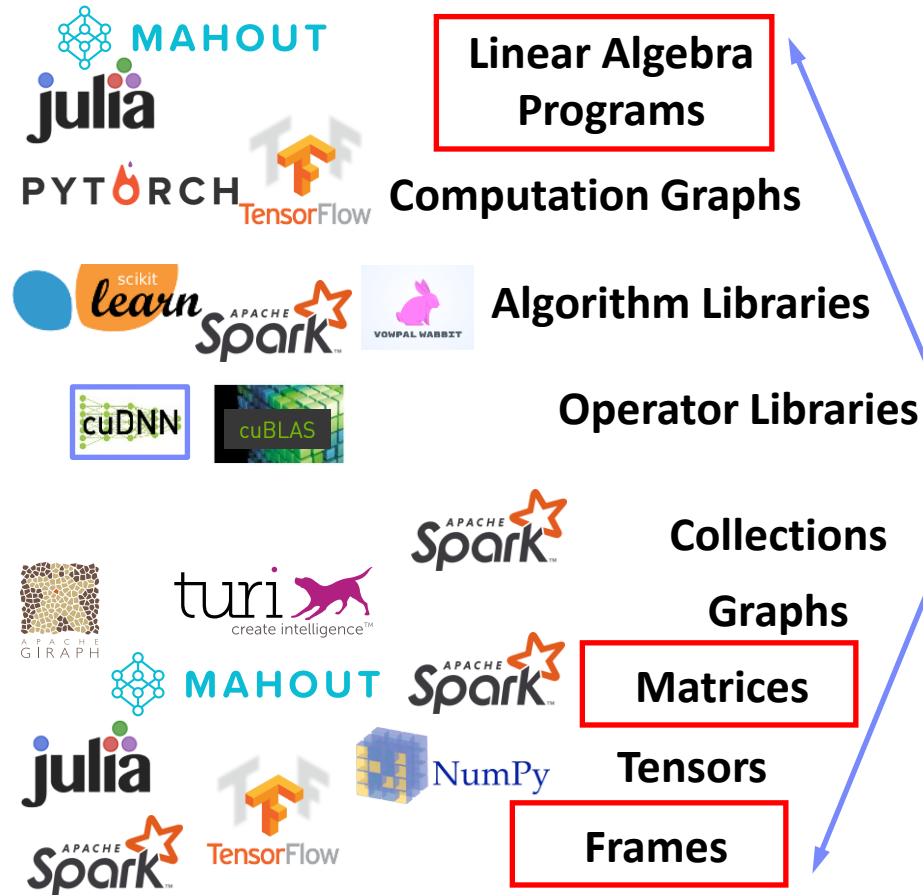
Landscape of ML Systems



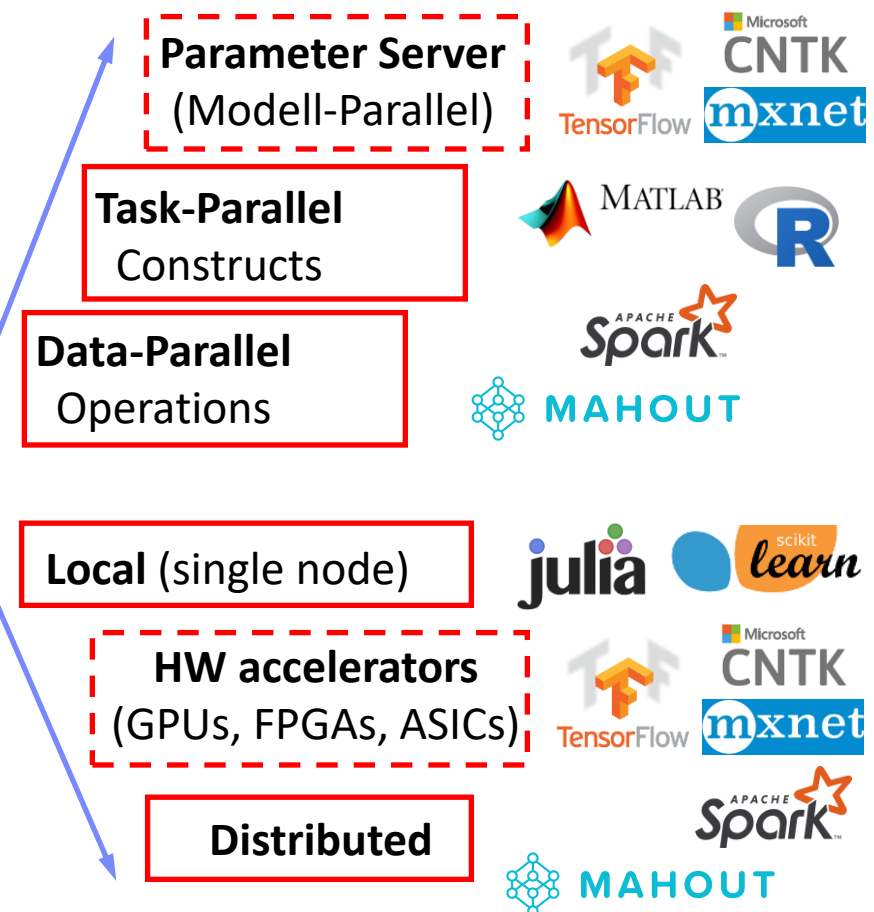
Landscape of ML Systems, cont. including Classification of SystemML/SystemDS



#1 Language Abstraction



#2 Execution Strategies



#4 Data Types

#3 Distribution

■ 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

```
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

```
SELECT A.i, B.j,  
       dot(A.row, B.col)  
FROM A, B;
```

Optimization w/ UDA

```
Init(state)  
Accumulate(state,data)  
Merge(state,data)  
Finalize(state,data)
```

Graph-based Systems

[Grzegorz Malewicz et al: **Pregel**:
a system for large-scale graph
processing. **SIGMOD 2010**,
SIGMOD 2020 Test of Time Award]



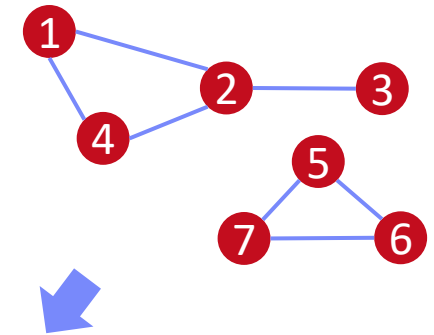
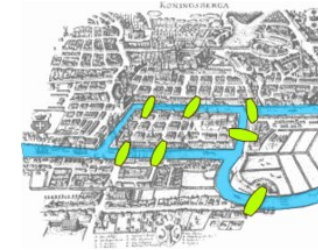
■ Google **Pregel**

- **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

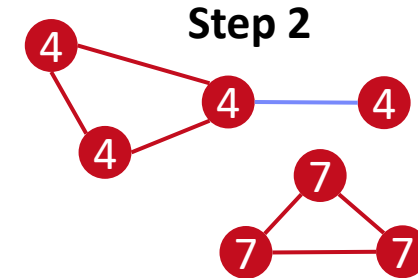
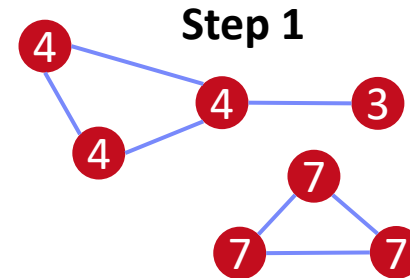
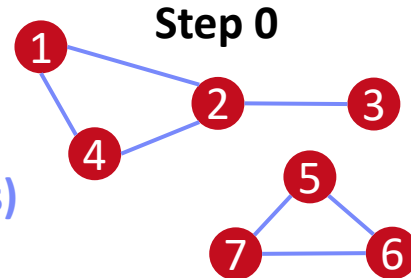
```
public abstract class Vertex {  
    public String getID();  
    public long superstep();  
    public VertexValue getValue();  
  
    public compute(Iterator<Message> msgs);  
    public sendMsgTo(String v, Message msg);  
    public void voteToHalt();  
}
```



2	[1, 3, 4]	Worker 1
7	[5, 6]	
4	[1, 2]	
1	[1, 2, 4]	
5	[6, 7]	Worker 2
3	[2]	
6	[5, 7]	

■ Example1: Connected Components

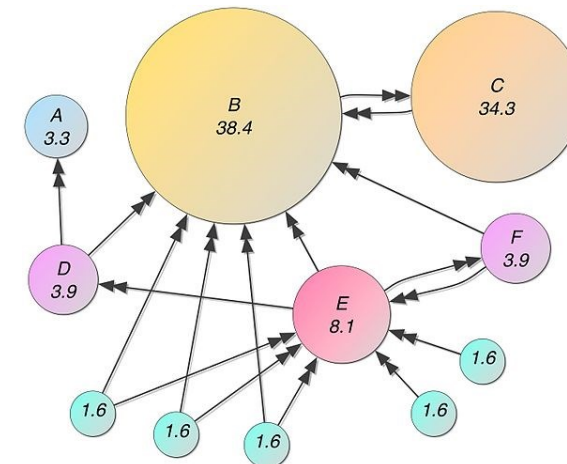
- Connected components of graph (subgraphs of connected nodes)
- Propagate $\max(\text{current}, \text{msgs})$ if $\neq$ current to neighbors, terminate if no msgs



Step 3
converged

■ Example 2: Page Rank

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



[Credit: <https://en.wikipedia.org/wiki/PageRank>]

- **Excursus: Graph Processing**
via **Sparse Linear Algebra**

- SystemDS'
components()

- SystemDS'
pageRank()



[Jure Leskovec, Anand Rajaraman,
Jeffrey D. Ullman: Mining of Massive
Datasets, **Stanford 2014**]

```
# 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;
}
```

■ 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)

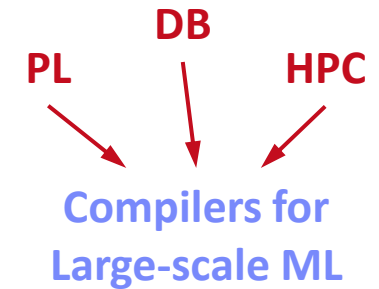
- Examples: [RIOT](#) [CIDR'09], [TensorFlow](#) [OSDI'16]
[Mahout Samsara](#) [MLSystems'16]
- Examples w/ control structures: [Weld](#) [CIDR'17],
[OptiML](#) [ICML'11], [Emma](#) [SIGMOD'15]

■ #3 Program Compilation (entire program)

- Examples: [SystemML](#) [PVLDB'16], [Julia](#)
[Cumulon](#) [SIGMOD'13], [Tupeware](#) [PVLDB'15]

Optimization Scope

```
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");
```



Linear Algebra Systems, cont.

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



Note: TF 2.0

■ Some Examples ...



```
X = read("./X");
y = read("./y");
p = t(X) %*% y;
w = matrix(0,ncol(X),1);
```

```
while(...) {
  q = t(X) %*% X %*% p;
  ...
}
```

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



```
var X = drmFromHDFS("./X")
val y = drmFromHDFS("./y")
var p = (X.t %*% y).collect
var w = dense(...)
X = X.par(256).checkpoint()
```

```
while(...) {
  q = (X.t %*% X %*% p)
    .collect
  ...
}
```

(Embedded DSL in Scala;
lazy evaluation)



```
# read via queues
sess = tf.Session()
# ...
w = tf.Variable(tf.zeros(...,
  dtype=tf.float64))
```

```
while ...:
  v1 = tf.matrix_transpose(X)
  v2 = tf.matmul(X, p)
  v3 = tf.matmul(v1, v2)
  q = sess.run(v3)
  ...
```

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

ML Libraries / Model Zoos



■ #1 Fixed Algorithm Implementations

- Often on top of existing linear algebra or UDF abstractions



Single-node Example (Python)

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

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

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

SparkML/
MLlib



Distributed Example (Spark 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()
    .fit(Xy)
val out reg.transform(Xy)
```

■ #2 Model Zoos / APIs

- Pre-trained models
- Hugging Face



(<https://huggingface.co/models>)

- YOLOv2 – v7
- PyTorch/TensorFlow

Model Zoos

PYTORCH





■ High-level DNN Frameworks

- Language abstraction for DNN construction and model fitting

- Examples:

Caffe, **Keras**

```
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))
...
```

```
opt = keras.optimizers.rmsprop(
    lr=0.0001, decay=1e-6)
```

```
# Let's train the model using RMSprop
model.compile(loss='cat..._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)
```



■ Low-level DNN Frameworks

- Examples: TensorFlow, MXNet, PyTorch, CNTK



LLM Offline and Online Inference



■ vLLM: LLM Inference and Serving

- Management of attention **key/value cache**
- Batching of incoming requests, eviction mechanisms
- **HuggingFace models** and **hardware accelerators**

[https://docs.vllm.ai/en/stable/getting_started/examples/basic.html#]

```
from vllm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]

# Create a sampling params object.
sampling_params = SamplingParams(
    temperature=0.8, top_p=0.95)

...
```

```
# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts (RequestOutput)
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
print("\nGenerated Outputs:\n" + "-" * 60)
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt:      {prompt!r}")
    print(f"Output:       {generated_text!r}")
    print("-" * 60)
```

■ DeepDive

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

[Jaeho Shin et al: Incremental Knowledge Base Construction Using DeepDive. **PVLDB 2015**]



■ Overton (Apple)

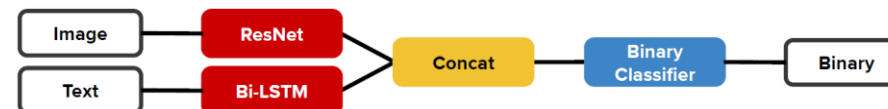
- Building, monitoring, improving ML pipelines
- High-level abstractions: **tasks** and **payloads**
- Data slicing, multi-task learning, data augmentation

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



■ Ludwig (Uber AI)

- **Data types** and **configuration files**
- Encoders, combiners, decoders
- **Example** “visual question answering”:



[Piero Molino, Yaroslav Dudin, Sai Sumanth Miryala: Ludwig: a type-based declarative deep learning toolbox. **CoRR 2019**]



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], TPCx-AI (09/2021)

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



■ HiBench (Intel)

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

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



■ GenBase

- Preprocessing and ML in array databases

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



■ SparkBench

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

[Dakshi Agrawal et al: **SparkBench** - A Spark Performance Testing Suite. **TPCTC 2015**]



Linear Algebra and DNN Benchmarks

AMLS'25 Project:
SSB Benchmark Implementation



- **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**
 - Open (diff model) vs closed divisions

[Anthony Thomas, Arun Kumar: A Comparative Evaluation of Systems for Scalable Linear Algebra-based Analytics. **PVLDB 2018**]



[Cody Coleman et al.: DAWNbench: An End-to-End Deep Learning Benchmark and Competition, **ML Systems Workshop 2017**]



ML
• Commons

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



[<https://mlcommons.org/en/training-normal-11/>,
<https://mlcommons.org/en/training-hpc-10/>]

DNN Benchmarks, cont.

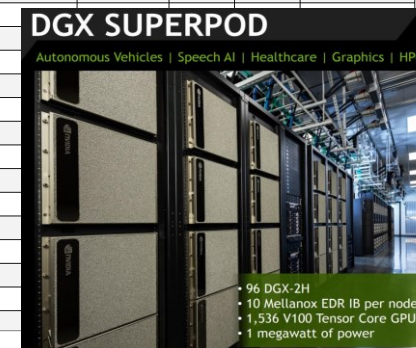


Closed Division Times										Benchmark results (minutes)							Details	Code	Notes	
#	Submitter	System	Processor	#	Accelerator	#	Software	Image classification	Object detection, light-weight	Object detection, heavy-wt.	Translation , recurrent	Translation , non-recur.	Recommendation	Reinforce-ment Learning						
								ImageNet	COCO	COCO	WMT E-G	WMT E-G	MovieLens-20M	Go						
								ResNet-50 v1.5	SSD w/ ResNet-34	Mask-RCNN	NMT	Transformer	NCF	Mini Go						
Available in cloud																				
0.6-1	Google	TPUv3.32			TPUv3	16	TensorFlow, TPU 1.14.1.dev	42.19	12.61	107.03	12.25	10.20	[1]			details	code	none		
0.6-2	Google	TPUv3.128			TPUv3	64	TensorFlow, TPU 1.14.1.dev	11.22	3.89	57.46	4.62	3.85	[1]			details	code	none		
0.6-3	Google	TPUv3.256			TPUv3	128	TensorFlow, TPU 1.14.1.dev	6.86	2.76	35.60	3.53	2.81	[1]			details	code	none		
0.6-4	Google	TPUv3.512			TPUv3	256	TensorFlow, TPU 1.14.1.dev	3.85	1.79		2.51	1.58	[1]			details	code	none		
0.6-5	Google	TPUv3.1024			TPUv3	512	TensorFlow, TPU 1.14.1.dev	2.27	1.34		2.11	1.05	[1]			details	code	none		
0.6-6	Google	TPUv3.2048			TPUv3	1024	TensorFlow, TPU 1.14.1.dev	1.28	1.21			0.85	[1]			details	code	none		
Available on-premise																				
0.6-7	Intel	32x 2S CLX 8260L	CLX 8260L	64			TensorFlow							[1]	14.43	details	code	none		
0.6-8	NVIDIA	DGX-1			Tesla V100	8	MXNet, NGC19.05	115.22						[1]		details	code	none		
0.6-9	NVIDIA	DGX-1			Tesla V100	8	PyTorch, NGC19.05		22.36	207.48	20.55	20.34		[1]		details	code	none		
0.6-10	NVIDIA	DGX-1			Tesla V100	8	TensorFlow, NGC19.05							[1]	27.39	details	code	none		
0.6-11	NVIDIA	3x DGX-1			Tesla V100	24	TensorFlow, NGC19.05							[1]	13.57	details	code	none		
0.6-12	NVIDIA	24x DGX-1			Tesla V100	192	PyTorch, NGC19.05			22.03				[1]		details	code	none		
0.6-13	NVIDIA	30x DGX-1			Tesla V100	240	PyTorch, NGC19.05		2.67					[1]		details	code	none		
0.6-14	NVIDIA	48x DGX-1			Tesla V100	384	PyTorch, NGC19.05				1.99			[1]		details	code	none		
0.6-15	NVIDIA	60x DGX-1			Tesla V100	480	PyTorch, NGC19.05					2.05		[1]		details	code	none		
0.6-16	NVIDIA	130x DGX-1			Tesla V100	1040	MXNet, NGC19.05	1.69						[1]		details	code	none		
0.6-17	NVIDIA	DGX-2			Tesla V100	16	MXNet, NGC19.05	57.87												
0.6-18	NVIDIA	DGX-2			Tesla V100	16	PyTorch, NGC19.05		12.21	101.00	10.94	11.04								
0.6-19	NVIDIA	DGX-2H			Tesla V100	16	MXNet, NGC19.05	52.74												
0.6-20	NVIDIA	DGX-2H			Tesla V100	16	PyTorch, NGC19.05		11.41	95.20	9.87	9.80								
0.6-21	NVIDIA	4x DGX-2H			Tesla V100	64	PyTorch, NGC19.05		4.78	32.72										
0.6-22	NVIDIA	10x DGX-2H			Tesla V100	160	PyTorch, NGC19.05					2.41								
0.6-23	NVIDIA	12x DGX-2H			Tesla V100	192	PyTorch, NGC19.05			18.47										
0.6-24	NVIDIA	15x DGX-2H			Tesla V100	240	PyTorch, NGC19.05		2.56											
0.6-25	NVIDIA	16x DGX-2H			Tesla V100	256	PyTorch, NGC19.05				2.12									
0.6-26	NVIDIA	24x DGX-2H			Tesla V100	384	PyTorch, NGC19.05				1.80									
0.6-27	NVIDIA	30x DGX-2H, 8 chips each			Tesla V100	240	PyTorch, NGC19.05		2.23											
0.6-28	NVIDIA	30x DGX-2H			Tesla V100	480	PyTorch, NGC19.05					1.59								
0.6-29	NVIDIA	32x DGX-2H			Tesla V100	512	MXNet, NGC19.05	2.59												
0.6-30	NVIDIA	96x DGX-2H			Tesla V100	1536	MXNet, NGC19.05	1.33												

DGX SUPERPOD

Autonomous Vehicles | Speech AI | Healthcare | Graphics |

- 96 DGX-2H
- 10 Mellanox EDR IB per node
- 1,536 V100 Tensor Core per node



MLPerf v0.6:
<https://mlperf.org/training-results-0-6/>,
 MLPerf v0.7:
<https://mlperf.org/training-results-0-7>
 ... MLPerf v2.1 (11/2022)

96 x DGX-2H = 96 * 16
 = 1536 V100 GPUs
 → ~ 96 * \$400K = **\$35M – \$40M**

[<https://www.forbes.com/sites/tiriasresearch/2019/06/19/nvidia-offers-a-turnkey-supercomputer-the-dgx-superpod/#693400f43ee5>]

■ MLBench

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

[Yu Liu et.al: MLBench: Benchmarking Machine Learning Services Against Human Experts. **PVLDB 2018**]



■ (Open Source) AutoML Benchmark

- 39 classification datasets, AUC metric, 10-fold CV
- Extensible metrics, OS **AutoML** frameworks, datasets

[Pieter Gijsbers et al.: An Open Source AutoML Benchmark. **Automated ML Workshop 2019**]



■ CleanML

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

[Peng Li et al: CleanML: A Benchmark for Joint Data Cleaning and Machine Learning, **ICDE 2021**]



■ Meta Worlds Benchmark

- **Meta-reinforcement** and **multi-task learning**
- 50 robotic tasks (e.g., get coffee, open window)

[Tianhe Yu et al: Meta-World: A Bench-mark and Evaluation for Multi-Task and Meta Reinforce-ment Learning, **CoRL 2019**]



■ Feature Type Inference

- Dirty/clean ML model training/test

[Vraj Shah et al.: Towards Benchmarking Feature Type Inference for AutoML Platforms, **SIGMOD 2021**]



Data-centric ML and other Directions



■ Excursus: ML-Commons Working Groups

<https://mlcommons.org/working-groups/>

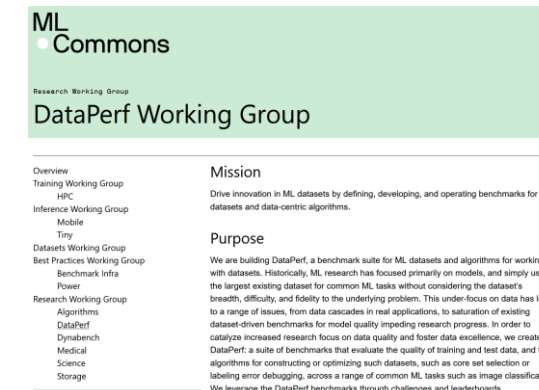
■ #1 DataPerf Working Group

- Renamed to Data-centric ML Research Working Group
- <https://mlcommons.org/en/groups/research-dataperf/>

■ #2 AI Safety Working Group

- v0.5: chatbots w/ typical, malicious, vulnerable users
- Defines 13 hazard categories
- <https://mlcommons.org/working-groups/ai-safety/ai-safety/>

■ And many more: e.g., datasets, metadata, medical domain



Introducing v0.5 of the AI Safety Benchmark from MLCommons

Bertie Vidgen¹, Adarsh Agrawal⁵³, Ahmed M. Ahmed^{2,9}, Victor Akinwande⁶⁰, Namir Al-Nuaimi⁵⁹, Najia Alfara⁶¹, Elie Alhajjar⁴, Lora Aroyo¹, Trupti Bavallatt⁶², Borhan Bili-Hameli⁶³, Kurt Bollacker¹, Rishi Bomassani¹, Maria Berra¹, Boston¹, Simón Campos⁶⁴, Kal Chakra¹, Canyu Chen³, Cody Coleman³, Zacharie Delpeyre Coudert¹, Leon Derczynski¹⁰, Debajyoti Dutta¹¹, Ian Eisenberg¹¹, James Edick¹⁵, Heather Frazer¹, Brian Fuller²⁸, Ram Gandikota¹⁰, Agasthya Gangavarapu¹⁵, Ananya Gangavarapu¹⁵, James Gealy²⁶, Rajat Ghosh¹¹, James Goe¹, Usman Gohar¹, Sujata Govwani¹, Scott A. Hale^{65,66}, Wieke Hutiri¹⁷, Joseph Marvin Imperial^{20,67}, Sargan Jandial¹¹, Nick Judd¹², Felix Juefei-Xu¹², Foutse Khomh¹², Bhavya Kailkhura¹², Hannah Rose Kirk¹⁴, Kevin Klym¹², Chris Koo¹², Michael Kuchnik¹², Shachi H. Kumar¹², Chris Lengwath¹², Bo Li¹², Zeyi Liao¹², Eileen Peters Long¹², Victor Lu¹², Yifan Mai¹², Priyanka Mary Mammen¹², Kelvin Manyek¹², Sean McGregor¹², Virendra Mehta¹², Shafee Mohammed¹², Emanuel Moss¹², Lama Nuchmany¹², Dinesh Jeyachally Naganna¹², Amin Nikanjam¹², Benmira Nushi¹², Luis Oala¹², Ifach Orr¹², Alicia Parrish¹², Cigdem Patlak¹², William Pietri¹², Forough Poursabzi-Sangleh¹², Eleonora Presani¹², Fabrizio Puletti¹², Paul Rötger¹², Saurav Sabay¹², Tim Santos¹², Nino Scherrer¹², Alice Schoenauer Sebarg¹², Patrick Schramowski¹², Abolfazl Shahbazi¹², Vin Sharma¹², Xudong Shen¹², Vamsi Srida¹², Leonard Tang¹², Davide Testuggine¹², Vithursan Thangarasa¹², Elizabeth Anne Watkins¹², Rebecca Weiss¹², Chris Welty¹², Tyler Wilbers¹², Adina Williams¹², Carole-Jean Wu¹², Poonam Yadav¹², Xianjun Yang¹², Yi Zeng¹², Wenhui Zhang¹², Fedor Zhdanov¹², Jiacheng Zhu¹², Percy Liang¹², Peter Mattson⁶⁸, Joaquin Vanschoren⁶⁹

¹MLCommons ²Stanford University ³Independent ⁴RAND ⁵Google Research ⁶Meta ⁷Beiza AI ⁸Illinois Institute of Technology ⁹Coactive AI ¹⁰NVIDIA ¹¹Nutanix ¹²Crede AI ¹³Qualcomm Technologies, Inc. ¹⁴Center for Security and Emerging Technology ¹⁵Juniper Networks ¹⁶Uthruva ¹⁷Caltech ¹⁸Yows State University ¹⁹Sony AI ²⁰University of Bath ²¹Adobe ²²New York University ²³Polyscience ²⁴Montreal ²⁵University of Oxford ²⁶Common Ground ²⁷PAIR, Meta ²⁸Intel Labs ²⁹Context Fund ³⁰University of Chicago ³¹The Ohio State University ³²UMass Amherst ³³Digital Safety Research Institute ³⁴University of Trento ³⁵Project Humanitai ³⁶Lawrence Livermore National Laboratory ³⁷MicroResearch ³⁸Dynabench ³⁹Microsoft ⁴⁰Bocconi University ⁴¹Patronus AI ⁴²DFKI & Hessian.AI ⁴³Intel

[<https://huggingface.co/papers/2404.12241>]

Summary & QA



- Data Science Lifecycle
- ML Systems Stack
- Language Abstractions
- ML System Benchmarks
- **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>

