TECHNISCHE UNIVERSITÄT BERLIN

HPI Lecture Series
on Database Research
WiSe 2023/24

System Infrastructure for Data-centric ML Pipelines

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

BIFOLD

About Me

- **Since 09/2022 TU Berlin**, Germany
  - University professor for Big Data Engineering (DAMS)
- **2018-2022 TU Graz**, Austria
  - BMK endowed chair for data management + research area manager
  - **Data management for data science** (DAMS), **SystemDS & DAPHNE**
- **2012-2018 IBM Research – Almaden**, CA, USA
  - Declarative large-scale machine learning
  - Optimizer and runtime of **Apache SystemML**
- **2007-2011 PhD TU Dresden**, Germany
  - Cost-based optimization of integration flows
  - Time series forecasting / in-memory indexing & query processing

DB group

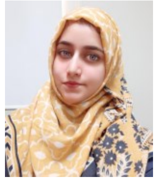
Motivation and Terminology

(ML) System Infrastructure for Data-centric ML Pipelines

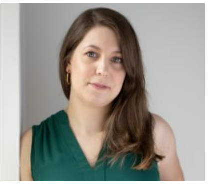
Data-centric ML Pipelines

Key observation: SotA data engineering/cleaning based on ML

Data Engineering



Top-K Cleaning Pipelines
[**SIGMOD'24a**]



Alignment of Multi-modal Data

I/O for Custom Data Formats
[**SIGMOD'23c**]

Parallel Feature Transformations
[**PVLDB'22**]

Data Preparation
(e.g., one-hot, bins)

FX

Data Integration & Data Cleaning

Data Programming & Augmentation

Model and Feature Selection

Hyper-parameter Tuning + CV

Hierarchical Composition as Library Functions on top of ML systems

`train()`

Model Training

X **y**

```
while(!converged) {
  … q = X %*% v …
}
```

`predict()`

Model Scoring

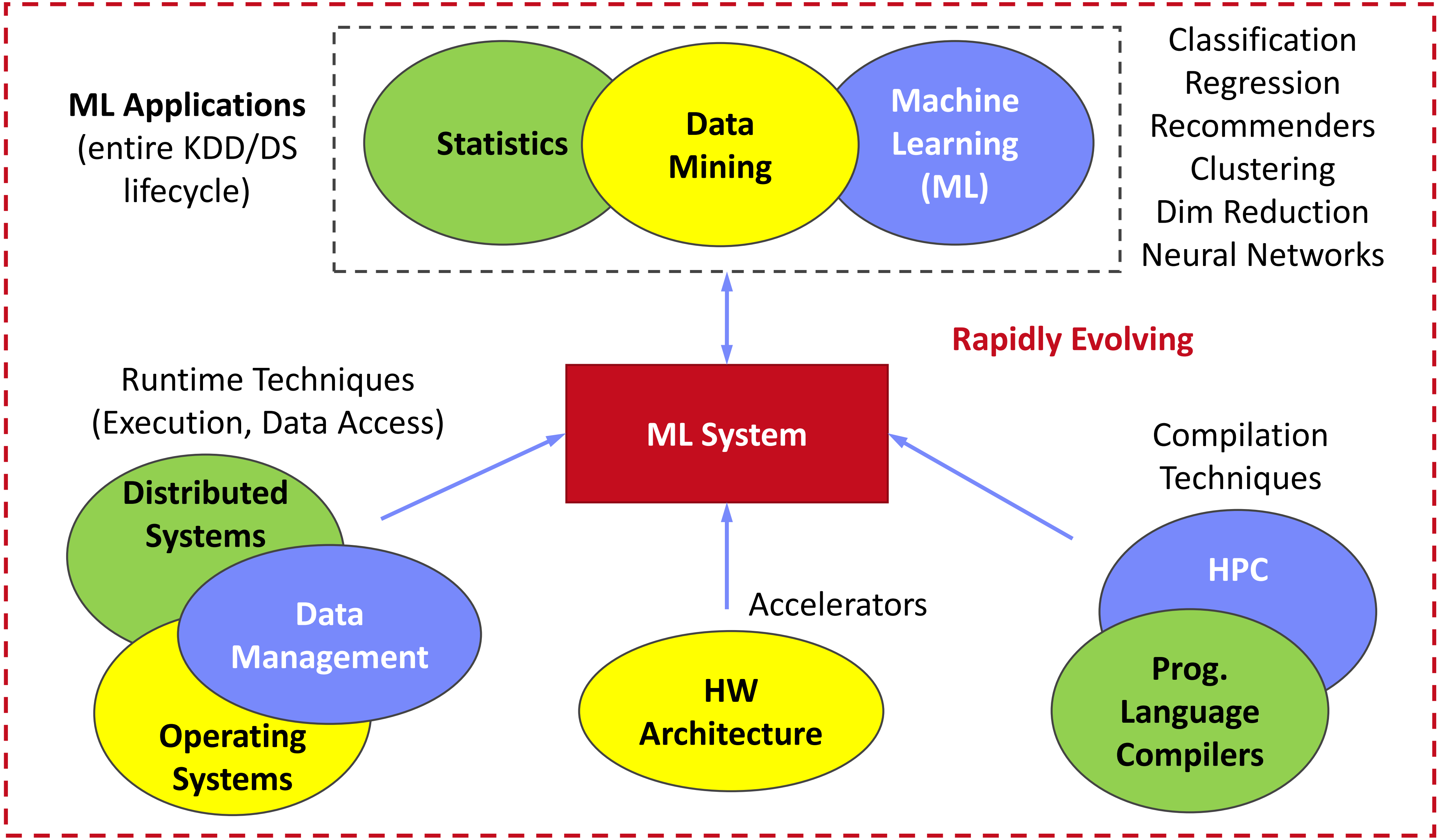
85%
Accuracy

SliceLine
[**SIGMOD'21c**]

Validation & Debugging

Deployment & Scoring

What is an ML System? (narrow vs broad scope)

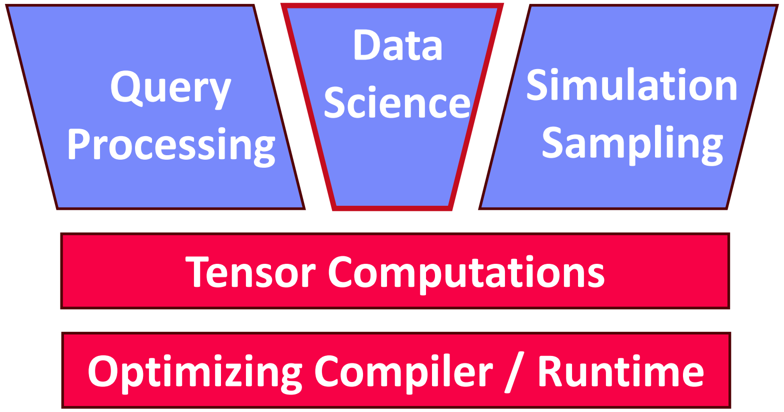


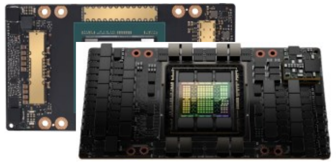
A Case for Optimizing Tensor Computations

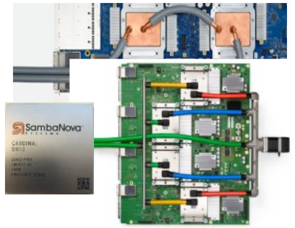
Optimizing Tensor Computations
From Applications to Compilation and Runtime Techniques

[SIGMOD'23 Tutorial]


- **#1 Simplicity**
  - Coarse-grained frame/matrix/tensor data structures and operations
  - Reduced system infrastructure complexity (boundary crossing)
- **#2 Reuse of Compiler/Runtime Techniques**
  - Focused work and reuse of commonly used compiler/runtime techniques
  - Generality over hand-crafted, specialized systems and algorithms
- **#3 Performance and Scalability**
  - Leverage HW Accelerators and distributed runtime backends
    ➔ Increasing specialization and rapid evolution
  - Homogeneous arrays and simple parallelization strategies

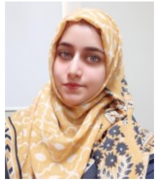






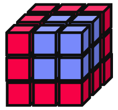

Build Libraries for Tensor Ops on HW X once and reuse

Data Science Lifecycle: Data Cleaning Pipelines [SIGMOD'24a]



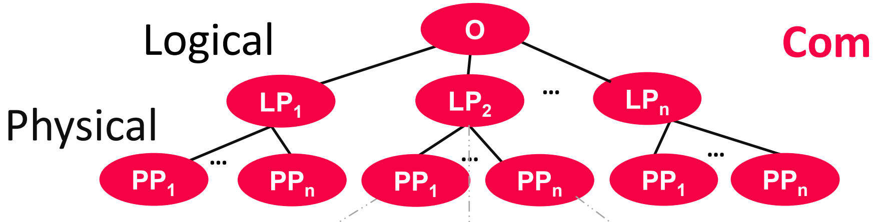
- **Automatic Generation of Cleaning Pipelines**
  - Library of robust, parameterized **data cleaning primitives**,
  - **Enumeration of DAGs** of primitives & **hyper-parameter optimization** (evolutionary, HB)

Data Samples



Target App

Rules/Objectives

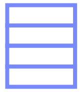


Data- and Task-parallel Computation

Outlier Detection → MVI → Deduplication → Resolve Mislabels

P_1: gmm → imputeFD → mergeDup → delML

P_n: outlierBySd → mice → delDup → voting



Top-k Pipelines

Debugging

University	Country
TU Graz	Austria
TU Graz	Austria
TU Graz	Germany
IIT	India
IIT	IIT
IIT	Pakistan
IIT	India
SIBA	Pakistan
SIBA	null
SIBA	null

Dirty Data

University	Country
TU Graz	Austria
TU Graz	Austria
TU Graz	Austria
IIT	India
IIT	India
IIT	India
IIT	India
SIBA	Pakistan
SIBA	Pakistan
SIBA	Pakistan

After **imputeFD(0.5)**

A	B	C	D
0.77	0.80	1	1
0.96	0.12	1	1
0.66	0.09	null	1
0.23	0.04	17	1
0.91	0.02	17	null
0.21	0.38	17	1
0.31	null	17	1
0.75	0.21	20	1
null	null	20	1
0.19	0.61	20	1
0.64	0.31	20	1

Dirty Data

A	B	C	D
0.77	0.80	1	1
0.96	0.12	1	1
0.66	0.09	17	1
0.23	0.04	17	1
0.91	0.02	17	1
0.21	0.38	17	1
0.31	0.29	17	1
0.75	0.21	20	1
0.41	0.24	20	1
0.19	0.61	20	1
0.64	0.31	20	1

After **MICE**

Data Science Lifecycle: SliceLine for Model Debugging

[SIGMOD'21b]

[**Credit:** sliceline, Silicon Valley, HBO]

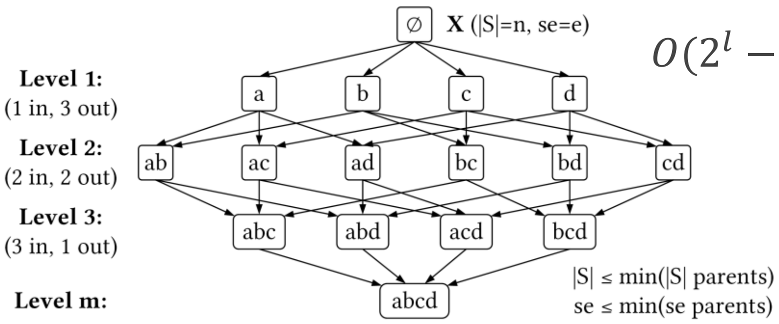
- **Problem Formulation**
  - **Intuitive slice scoring function**
  - Exact **top-k slice finding**
  - $|S| \geq \sigma \wedge sc(S) > 0,\ \alpha \in (0,1]$

$$sc = \alpha\left(\frac{\bar{e}(S)}{\bar{e}(X)} - 1\right) - (1-\alpha)\left(\frac{|X|}{|S|} - 1\right)$$

$$= \alpha\left(\frac{|X|}{|S|} \cdot \frac{\sum_{i=1}^{|S|} es_i}{\sum_{i=1}^{|X|} e_i} - 1\right) - (1-\alpha)\left(\frac{|X|}{|S|} - 1\right)$$

slice error **slice size**

- **Properties & Pruning**
  - Monotonicity of slice sizes, errors
  - **Upper bound sizes/errors/scores**
    → pruning & termination



$$O(2^l - \sum_{j=1}^{m} 2^{d_j} + l + m)$$

- **Linear-Algebra-based Slice Finding**
  - Recoded/binned matrix **X**, error vector e
  - **Vectorized implementation in linear algebra** (join & eval via sparse-sparse matmult)
  - Local and distributed task/data-parallel execution

Candidate Slices

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Data

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 & 0 \\ 0 & 2 & 0 \\ 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 1 \end{bmatrix}$$

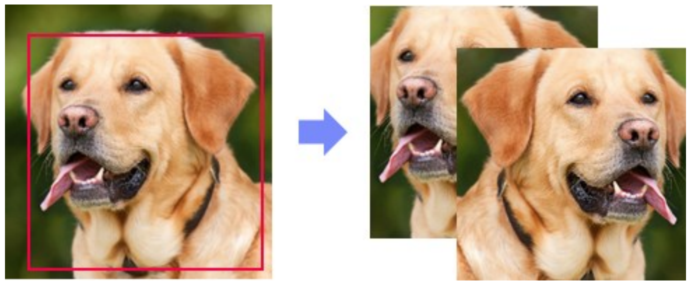
== Level

BIFOLD

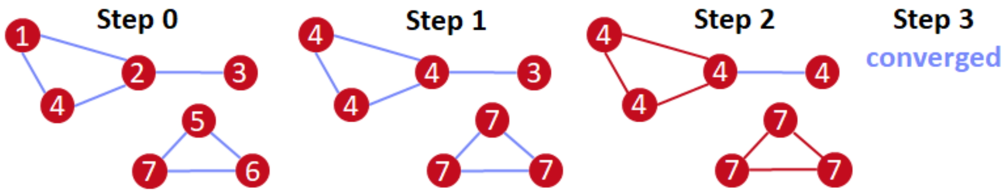
Data Science Lifecycle: Other Examples

- **Data Augmentation**
  - Augment training data by **synthetic labeled data**
  - **#1: Movement/selection** (translation, rotation, reflection, cropping)
  - **#2: Distortions** (stretching, shearing, lens distortions, color, mixup)

AlexNet



- **Graph Processing**
  - **Graphs are sparse matrices**
  - Connected components, page rank, shortest path



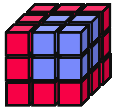
- **ML Algorithms**
  - Clustering, dimensionality reduction, matrix factorization and completion
  - Linear models, tree-based models, deep neural networks
- **Fairness and Explainability**
  - Group fairness constraints and monotonicity
  - Locally weighted regression

Clean Mappings to Linear Algebra Operations

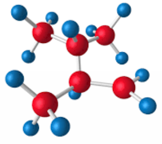
System Infrastructure for Data-centric ML Pipelines

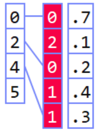
Need for Data Independence

#1 Data Representations



graph





dense

sparse

compressed

Sparsity Exploitation from Algorithms to HW

#2 Data Placement

Local vs **distributed**

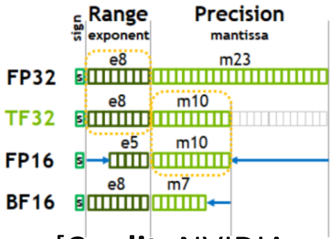
CPUs/ NUMA

GPUs

FPGAs/ ASICs

#3 Data (Value) Types

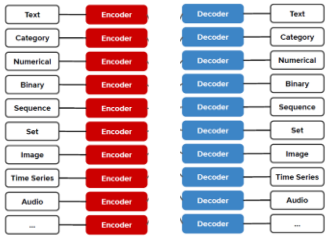
FP32, FP64, INT8, INT32, INT64, UINT8, BF16, TF32, FlexPoint



[**Credit:** NVIDIA A100 Whitepaper]

#4 Data Modalities

Text, Structured, Time Series, Image, Speech



[**Credit:** Uber AI Ludwig paper]

- **Data Independence**
  - "the independence of application programs […] from growth in data types and changes in data representations"

$$\frac{\Delta env}{\Delta t} \gg \frac{\Delta app}{\Delta t}$$

[E. F. Codd: A Relational Model of Data for Large Shared Data Banks. Comm. ACM 13(6), **1970**]

[J. Hellerstein: **2005** https://dsf.berkeley.edu/cs262/SystemR-annotated.pdf]

#1 Apache SystemDS [https://github.com/apache/systemds]

APIs: Command line, JMLC, Python
Spark MLContext, Spark ML,
(Scalable Algorithms + Primitives)

DML Scripts

Language

Compiler

Runtime

Write Once,
Run Anywhere

[SIGMOD’15,’17,’19,’21abc,’23abc,’24a]
[PVLDB’14,’16ab,’18,’22]
[ICDE’11,’12,’15]
[CIDR’17,’20]
[VLDBJ’18]
[CIKM’22]
[DEBull’14]
[PPoPP’15]

Apache SystemML™

07/2020 Renamed to **Apache SystemDS**
05/2017 Apache Top-Level Project
11/2015 Apache Incubator Project
08/2015 Open Source Release

In-Memory Single Node
(scale-up)

Hadoop or Spark Cluster
(scale-out)

Federated
(LA progs, PS)

Others:
Netezza
Apache Flink

In-Progress:
GPU

since 2014/16

Java

since 2012

hadoop

since 2010/11

APACHE Spark

since 2015

exdra

since 2019

Language Abstractions and APIs

Data Independence + Impl-Agnostic Ops
➔ **"Separation of Concerns"**

- **Example:** Stepwise Linear Regression

User Script

```
X = read('features.csv')
Y = read('labels.csv')
[B,S] = steplm(X, Y,
  icpt=0, reg=0.001)
write(B, 'model.txt')
```

Built-in Functions

```
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
    } }
    # add best to Xg
    # (AIC)
} }
```

Feature Selection

```
m_lm = function(...) {
  if( ncol(X) > 1024 )
    B = lmCG(X, y, ...)
  else
    B = lmDS(X, y, ...)
}
```

ML Algorithms

```
m_lmCG = function(...) {
  while( i<maxi&nr2>tgt ) {
    q = (t(X) %*% (X %*% p))
      + lambda * p
    beta = ... }
}
```

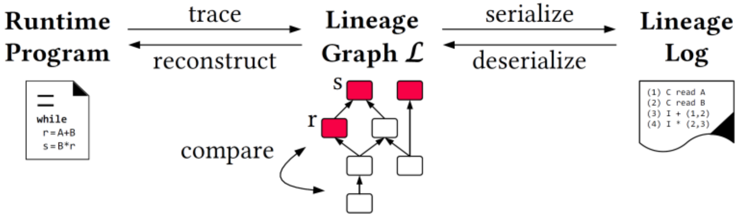
```
m_lmDS = function(...) {
  l = matrix(reg,ncol(X),1)
  A = t(X) %*% X + diag(l)
  b = t(X) %*% y
  beta = solve(A, b) ...}
```

Linear Algebra Programs

Facilitates optimization across data science lifecycle tasks

#2 Multi-level Lineage Tracing & Reuse [CIDR'20, SIGMOD'21a]


- **Lineage as Key Enabling Technique**
  - Trace lineage of ops (incl. non-determinism), dedup for loops/funcs
  - Model versioning, data reuse, incr. maintenance, autodiff, debugging

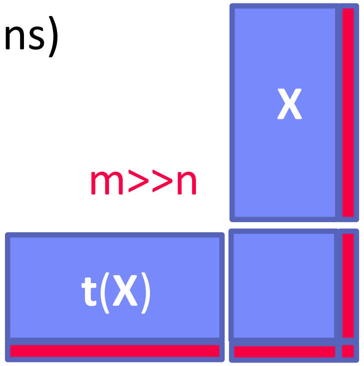


- **Full Reuse of Intermediates**
  - Before executing instruction, probe output lineage in cache `Map<Lineage, MatrixBlock>`
  - Cost-based/heuristic caching and eviction decisions (compiler-assisted)

```
for( i in 1:numModels )
  R[,i] = lm(X, y, lambda[i,], ...)
```

```
m_lmDS = function(...) {
  l = matrix(reg,ncol(X),1)
  A = t(X) %*% X + diag(l)
  b = t(X) %*% y
  beta = solve(A, b) ...}
```

- **Partial Reuse of Intermediates**
  - **Problem:** Often partial result overlap
  - Reuse partial results via dedicated rewrites (compensation plans)
  - Example: steplm

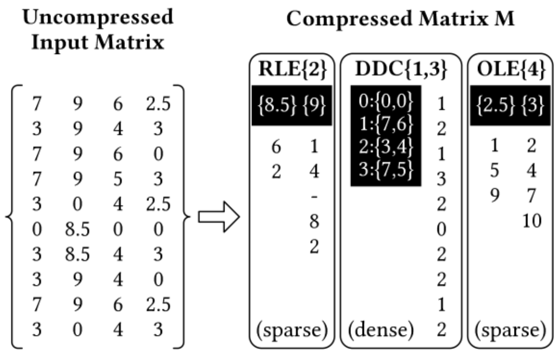



```
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
    } }
    # add best to Xg (AIC)
} }
```

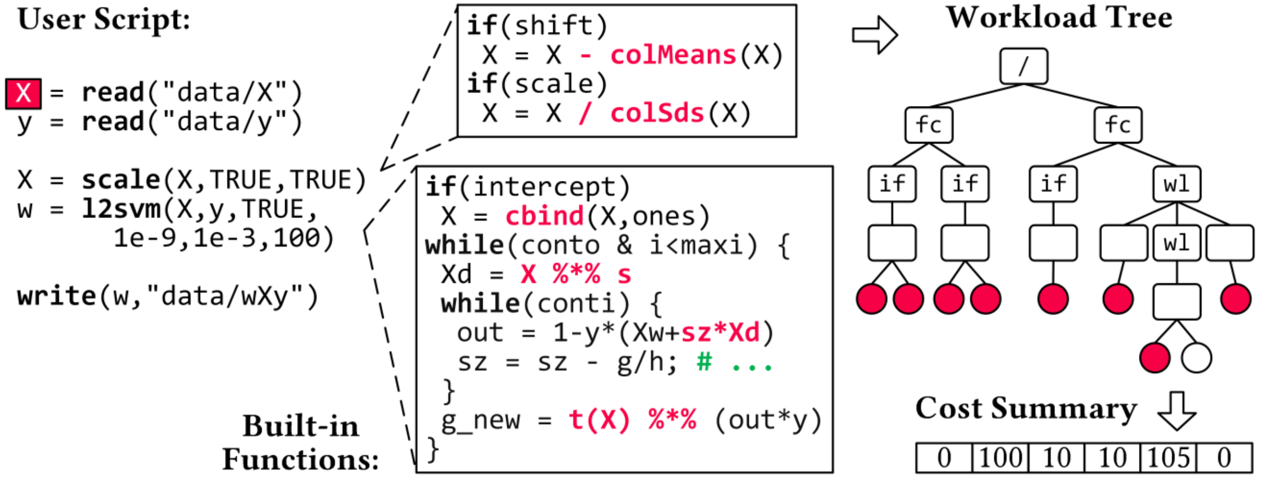
- **Next Steps:** multi-backend, unified mem mgmt

#3 Compressed Linear Algebra Extended [PVLDB'16a, VLDBJ'18, SIGMOD'23a]

- **Lossless Matrix Compression**
  - Improved general applicability (adaptive compression time, new compression schemes, new kernels, intermediates, workload-aware)
  - Sparsity → Redundancy exploitation (data redundancy, structural redundancy)



- **Workload-aware Compression**
  - Workload summary → compression
  - Compressed Representation → execution planning



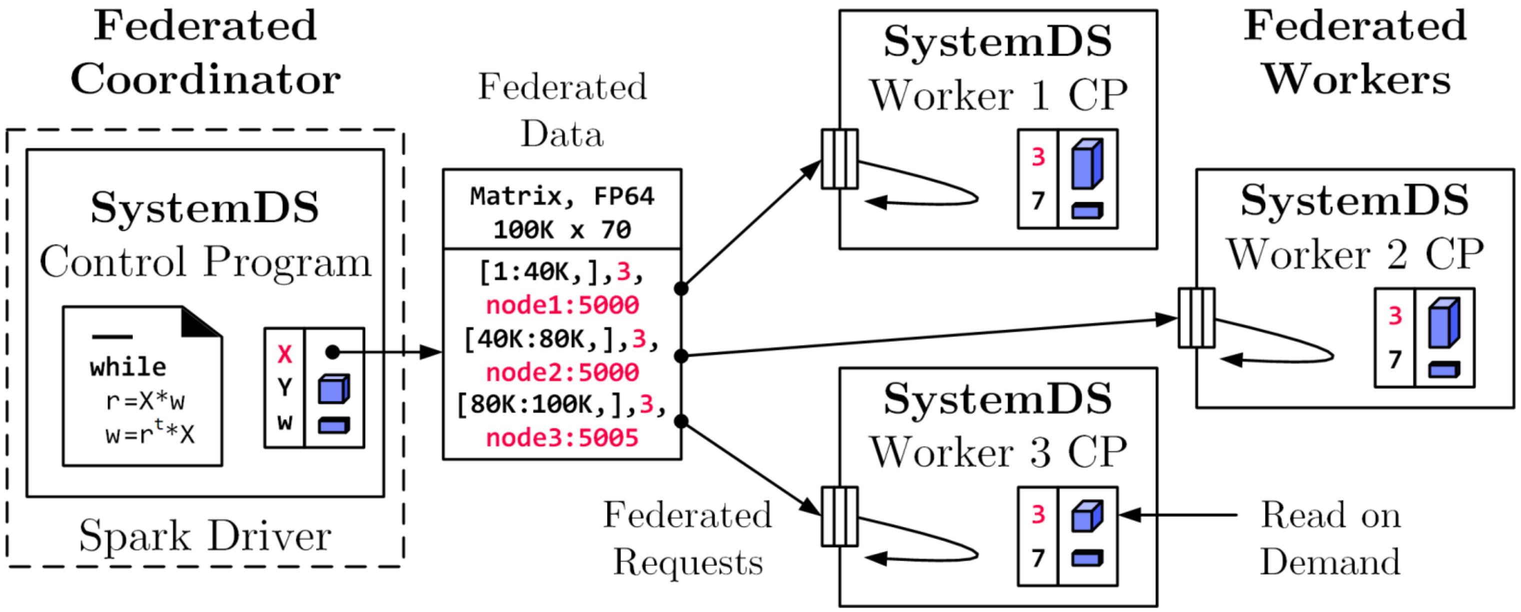
- **Next Steps**
  - Frame compression, compressed I/O
  - Compressed feature transformations
  - Morphing of compressed data

#4 Federated Learning in SystemDS

[SIGMOD'21, CIKM'22]

- **Federated Backend**
  - **Federated data** (matrices/frames) as meta data objects
  - **Federated linear algebra**, (and **federated parameter server**)

```
X = federated(addresses=list(node1, node2, node3),
  ranges=list(list(0,0), list(40K,70), ..., list(80K,0), list(100K,70)));
```



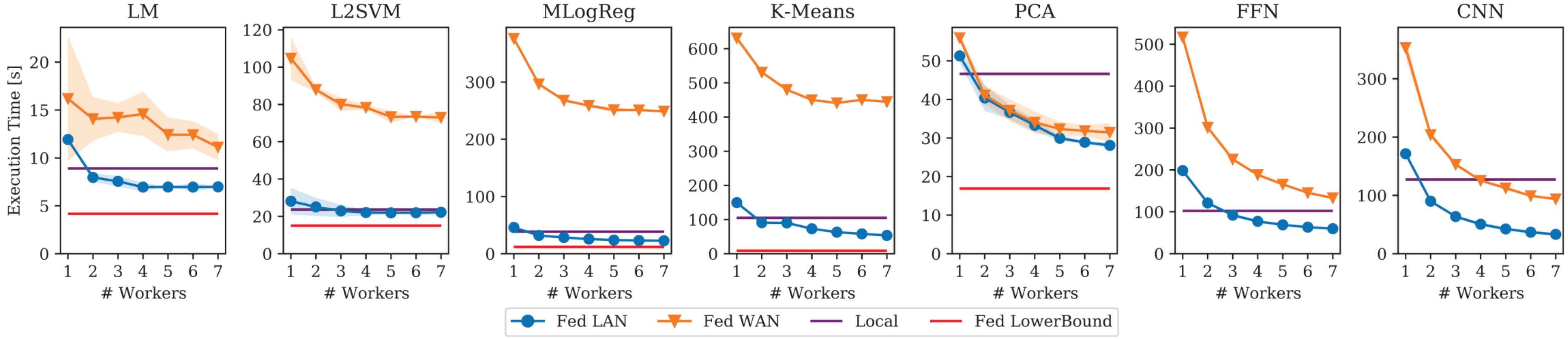
Federated Requests:
`READ, PUT, GET, EXEC_INST, EXEC_UDF, CLEAR`

➔ **Design Simplicity:**
(1) reuse instructions
(2) federation hierarchies

#4 Federated Learning in SystemDS – Experiments

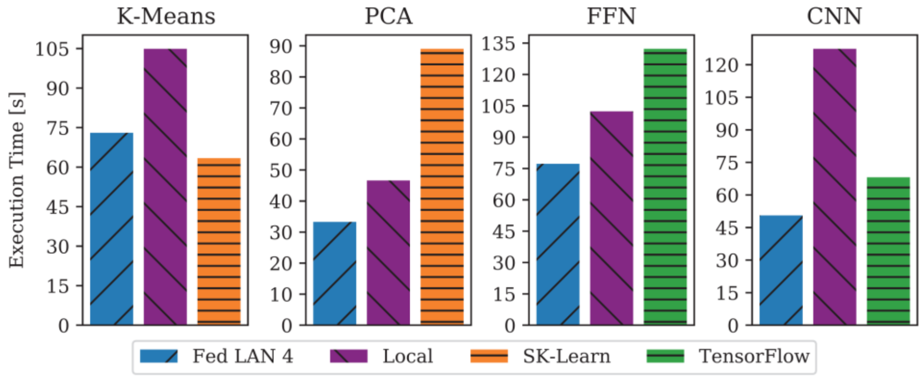
Reproducible Results

OPEN ACCESS



- **Workloads and Baselines**
  - **LM:** linear regression, lmCG
  - **L2SVM:** l2-regularized SVM
  - **MLogReg:** multinomial logreg
  - **K-Means:** Lloyd's alg. w/ K-Means++ init
  - **PCA:** principal component analysis
  - **FFN:** fully-connected feed-forward NN
  - **CNN:** convolutional NN

Comparisons w/ **Scikit-learn** and **TensorFlow**



#5 Fine-grained Device Placement in DAPHNE

[CIDR'22, NoDMC'23]

- **Design Principles**
  - Towards Integrated Data Analysis Pipelines
  - Abstract Frame and Matrix Operations
  - Open and Extensible Infrastructure

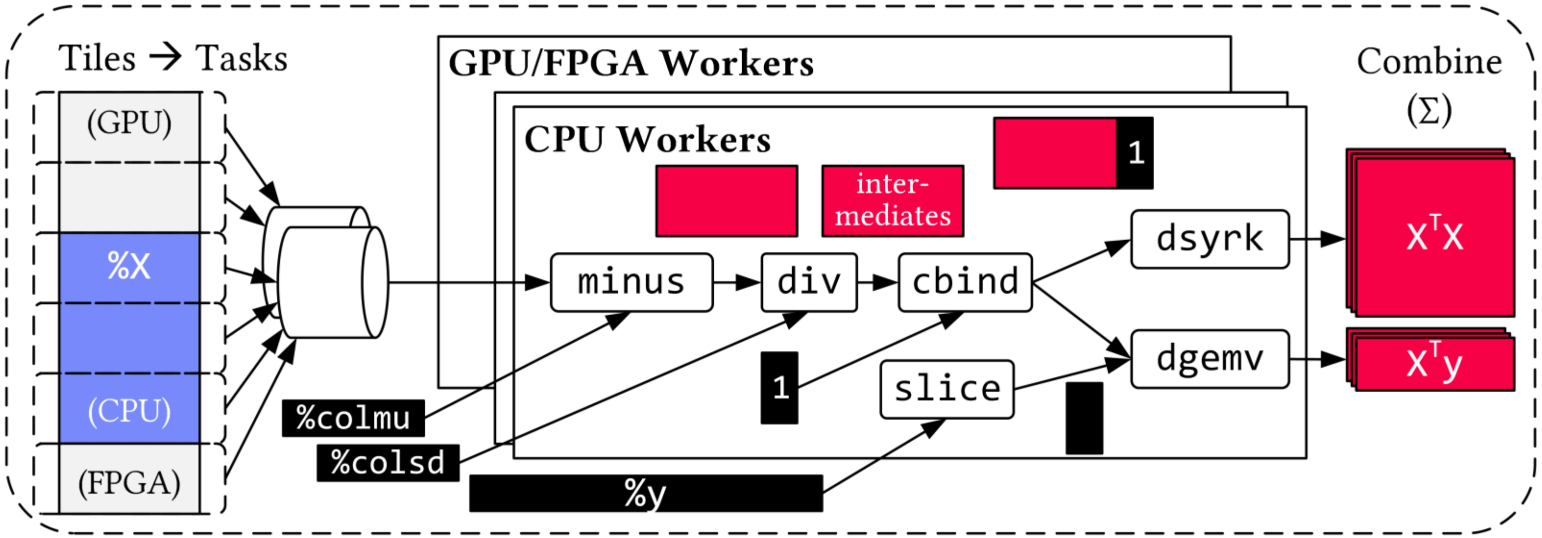
➔ DM + HPC + ML

DAPHNE

- **Vectorized (Tiled) Execution Engine**

Federated Data
➔ Multi-device Data

```
(%9, %10) = fusedPipeline1(%X, %y, %colmu, %colsd) {
```

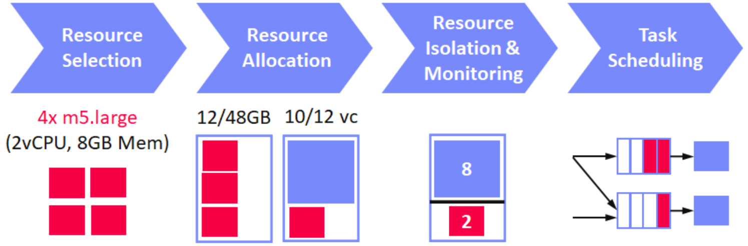


What's Next:
Towards Holistic Redundancy Exploitation

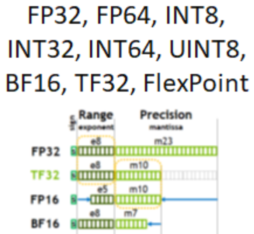
[rejected ERC consolidator grant proposal 2023]

Redundancy-exploiting Techniques for data-centric ML Pipelines

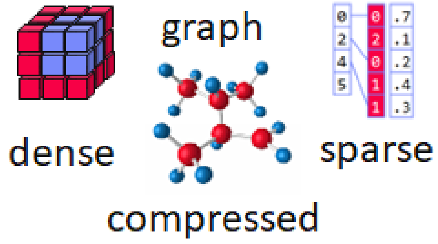
- **Resource Allocation and Elasticity**



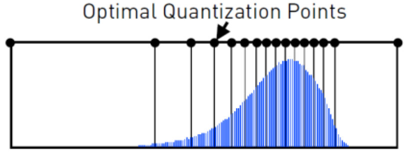
- **Data Sampling and Composition**
  - Sampling, distillation, augmentation-as-a-kernel, factorization



[**Credit:** NVIDIA A100 Whitepaper]



- **Sparsity Exploitation**
  - Algorithms, op pipelines, data/weights, kernels, HW

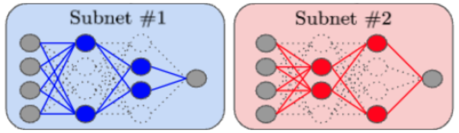


[**Credit:** Ce Zhang]

- **Lossy and Lossless Compression**

- **Weight Pruning and Connection Sampling**

Isolated Application, Exploration, and Tuning; Trial-and-Error Process



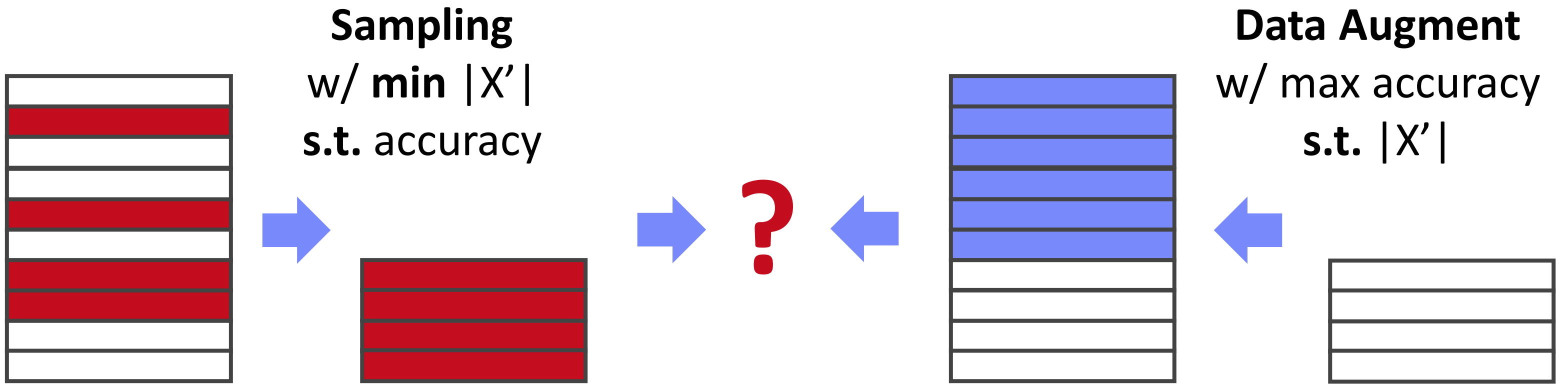
[**Credit:** Chris Jermaine]

LAURYN: Towards Holistic Redundancy Exploitation

- **Overall Approach**
  - End-to-end learning of a holistic multiplexing of redundancy-exploiting techniques
  - **Lossy decisions learned at algorithm level** (sampling, sparsification, lossy compression), combined with **lossless sparsity exploitation and compression at systems level**

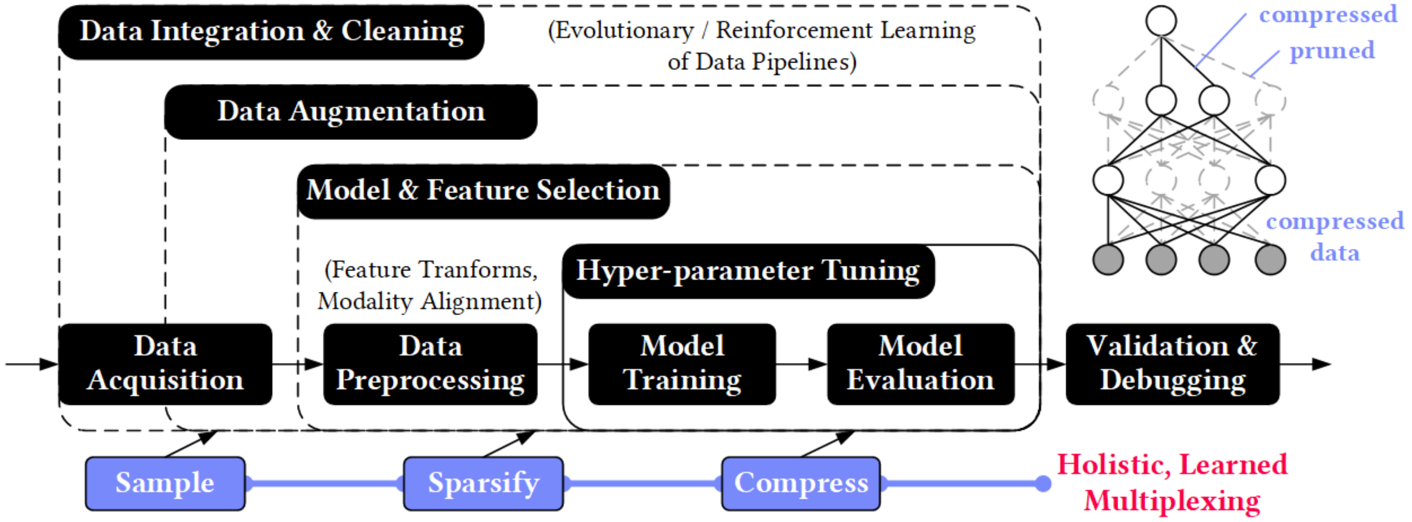
$$W' = arg \min_{W} \quad E_D(W) + \lambda \cdot R(W) + \cdots + \underbrace{\lambda_S \cdot \Sigma_{i=1}^{n} (W_i \neq 0)}_{\#\text{non-zeros}} + \underbrace{\lambda_C \cdot |W|}_{\#\text{distinct}}$$

- **Currently Ongoing Sub-projects**
  - **#1** Learned **sampling** and **data augmentation**
  - **#2** Learned sparsification and lossy quantization
  - ➔ How to combine these learning strategies?



LAURYN: Towards Holistic Redundancy Exploitation, cont.

- **Overall Goal:**
  Learned Multiplexing



- **Multi-objective Optimization with Hierarchical Multiplexing**

```
while(!convergedOuter) {
  X1 = sample(X, ...)
  while(!convergedInner) {
    X2 = compress(X2, |X|)
    … q = X2 %*% w …
  }
}
```

(proxy models sufficient?)

Automatic Redundancy Exploitation
(**foundational advancements** for sparsity/error estimators, new sparse/compressed data types and kernels, workload awareness)

Conclusions & QA

Thanks

- **#1 Data-centric ML Pipelines**
  - Increasingly complex, composite ML pipelines
  - State-of-the-art data engineering methods based on ML
  - Partial **resource, operational, and data redundancy**

→ **Optimizing Compiler and Runtime Infrastructure**

- **#2 Holistic Redundancy Exploitation (LAURYN)**
  - **Learned multiplexing of redundancy-exploiting techniques** (application and parameterization)
  - Robust ML system integration for **end-to-end improvements**

→ **Learn Lossy Decisions of Redundancy Exploitation**

- **TU Berlin – Big Data Engineering (DAMS Lab)**
  - **#1 Integrated Data Analysis Pipelines** (specialized for workload & HW)
  - **#2 Automatic Data Reorganization** (specialized for data characteristics)
  - **#3 Data Engineering and Model Debugging** (specialized for domain)
  - **#4 Data Platforms, Federated and Cloud Infra** (specialized deployment)

➔ Needs appropriate **Abstractions** and inter-disciplinary **Collaborations**

https://github.com/apache/systemds
https://github.com/daphne-eu/daphne