Data Integration and Large Scale Analysis

Slides credit: Matthias Boehm - Shafaq Siddiqi

12- Distributed ML Systems

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Lucas Iacono. PhD. - 2025

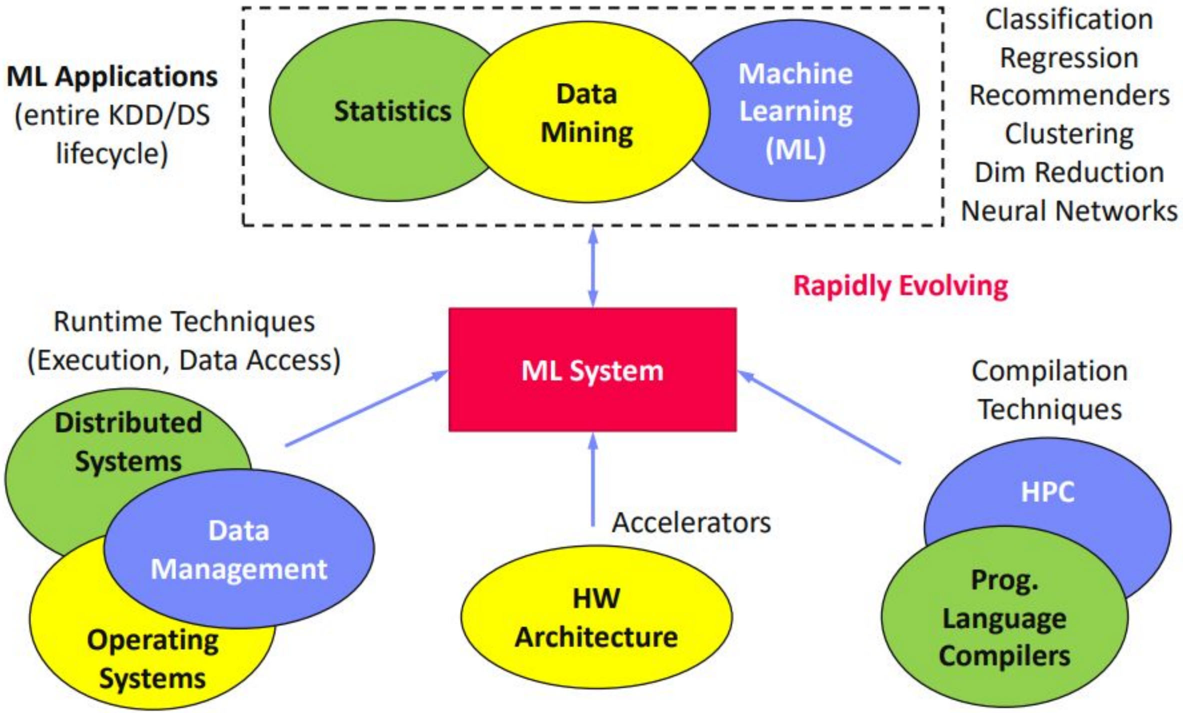
ISDS

Agenda

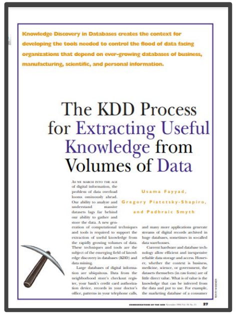
- Landscape of ML Systems
- Distributed Parameter Servers
- Large Language Models at HPC
- Q&A and Exam Preparation

Landscape of ML Systems

What is an ML System?



From KDD to the AI Lifecycle


Knowledge Discovery in Databases creates the context for developing the tools needed to control the flood of data facing organizations that depend on ever-growing databases of business, manufacturing, scientific, and personal information.

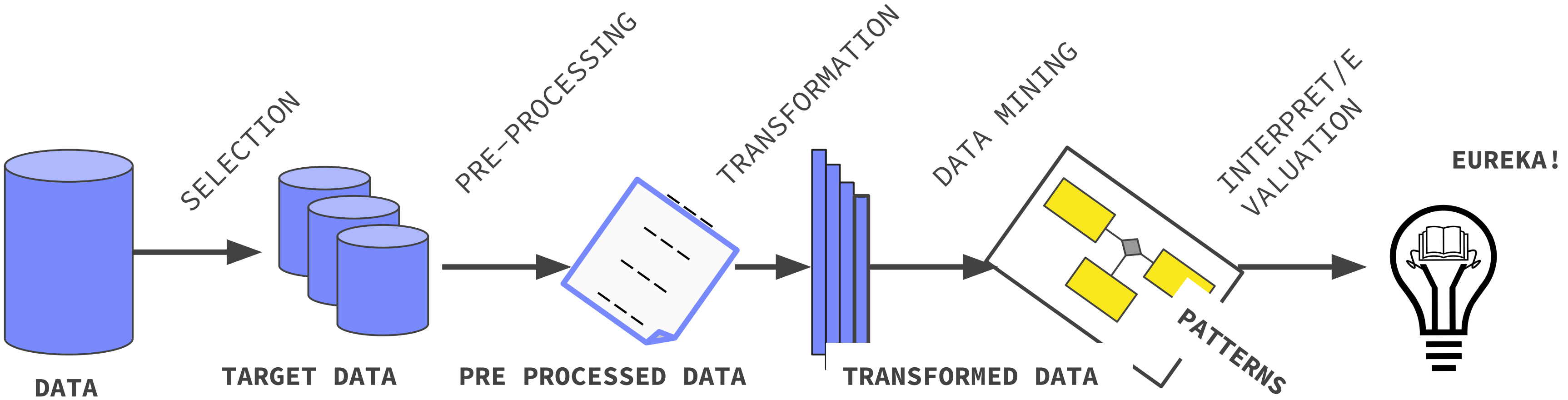
The KDD Process for Extracting Useful Knowledge from Volumes of Data

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth

Classic KDD (Knowledge Discovery in Databases)

Descriptive (association rules, clustering) and **predictive**

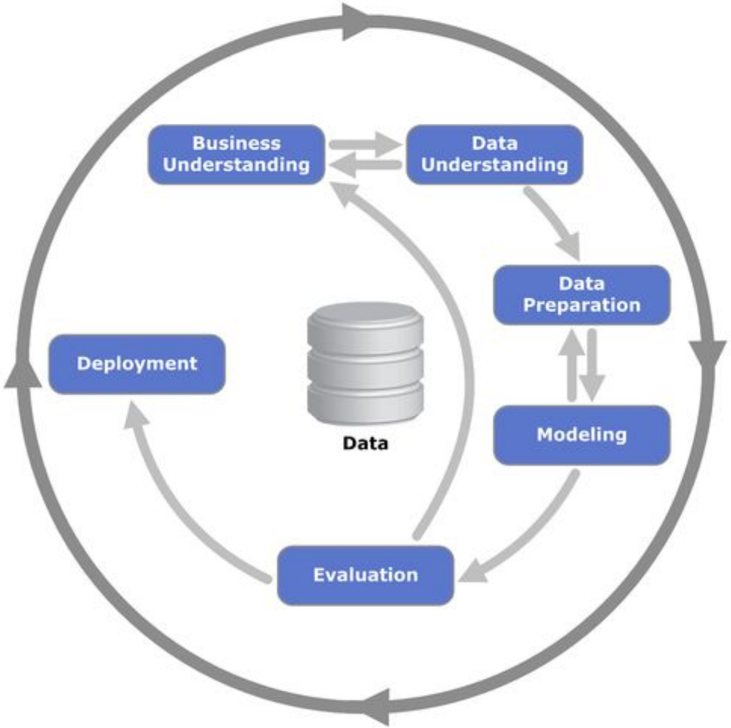
Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, *39*(11), 27-34.



From KDD to the AI Lifecycle

CRISP-DM (Cross-Industry Standard Process for Data Mining)

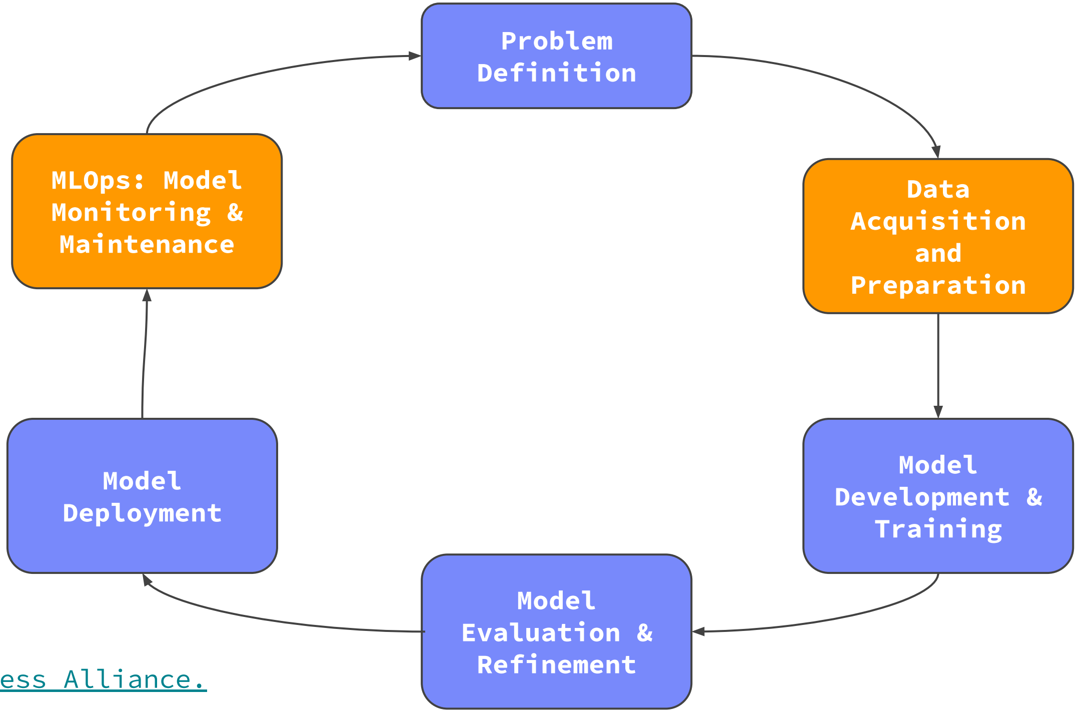
What's new? Business Understanding and Deployment **(A business perspective)**



Source: Statistik Dresden

From KDD to the AI Lifecycle

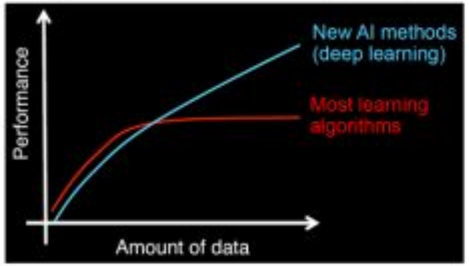
AI Lifecycle



[Ref: Jeffrey Saltz. Data Science Process Alliance.]

Driving Factors for ML

Credit: Andrew Ng’ 14



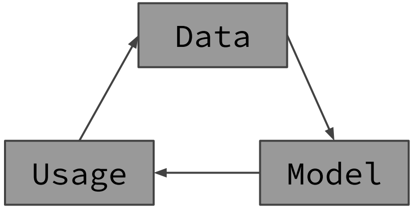
Improved Algorithms and Models

- 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, data programming/augmentation

Feedback Loop



Driving Factors for ML

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

Training

ML Apps & Algorithms

Supervised, unsupervised, RL, libs, AutoML

Stack of ML Systems

Training

ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation

Stack of ML Systems

Training

ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC

Stack of ML Systems

Training

ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)

Stack of ML Systems

Training

Layer	Examples
ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)
Data Representations	Dense & sparse tensor/matrix; compress, partition, cache

Stack of ML Systems

Training

ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)
Data Representations	Dense & sparse tensor/matrix; compress, partition, cache
HW & Infrastructure	CPUs, NUMA, GPUs, FPGAs, ASICs, RDMA, SSD/NVM

Stack of ML Systems

Training

Deployment & Scoring

Validation & Debugging

Hyper-parameter tuning

Model and Feature Selection

Data Programming and Augmentation

Data Preparation (e,g, one-hot)

Data Integration & Data Cleaning

ML Apps & Algorithms	Supervised, unsupervised, RL, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)
Data Representations	Dense & sparse tensor/matrix; compress, partition, cache
HW & Infrastructure	CPUs, NUMA, GPUs, FPGAs, ASICs, RDMA, SSD/NVM

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

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

Accelerators (GPUs, FPGAs, ASICs)

NVIDIA Volta (“tensor cores” for 4x4 mm -> 64 2B FMA instruction)

- **Tensor cores**
  - Processing units introduced in Volta architecture
  - Accelerate matrix **multiplications** and **convolutions**
- **4x4 mm**
  - Each **tensor** can multiply **two 4x4 matrices**.
- **FMA (Fused Multiply-Add) instruction**
  - **Multiplication** of two numbers and **directly adds** the result to **another number** in a **single step.**
- **2B** (2-byte). Each value being multiplied (e.g. weights and activations) is 16 bits (half-precision) **-> Faster** computation and **less memory** bandwidth

Accelerators (GPUs, FPGAs, ASICs)

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 1B FMA), NVIDIA DLA, Intel NNP

Data Representation

Apps
Lang
Faults
Exec
Data
HW

ML- vs DL-centric Systems

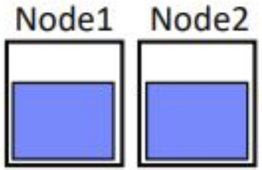
- **ML:** dense and sparse matrices or tensors, different sparse formats (CSR, CSC, COO), frames (heterogeneous data formats)
- **DL:** mostly **dense tensors, embeddings (dense representations of words or tokens)** for NLP

vec(Vienna) - vec(Austria)
vec(Italy) = vec(Rome)

Data Representation

Data-Parallel Operations for ML

- Distributed matrices:
    - RDD <MatrixIndexes,MatrixBlock > (Spark)
- Data properties: distributed caching, partitioning, compression

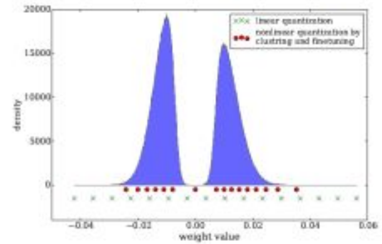


Data Representation

Lossy Compression -> Acc/Perf-Tradeoff

- Sparsification (reduce non-zero values)
- Quantization (reduce value domain 32 bit floats to 8-bit integers)
- New data types: Intel Flexpoint (mantissa, exp)
    - E.g: a 32 F Bits Integer can be represented as a 8-bit **mantissa** with a **shared exponent**

[**Credit:** Song Han’16]



Execution Strategies

Apps
Lang
Faults
Exec
Data
HW

Batch Algorithm:

Compute large datasets in **blocks** (not one data p/time)

- **Data-parallel** Split data into chunks -> dist + compute
- **Task-parallel** Divide workload into tasks -> dist + compute
- **Different** strategies to implement **physical operators** (e.g. “sum”) according to the architecture (local, istributed)

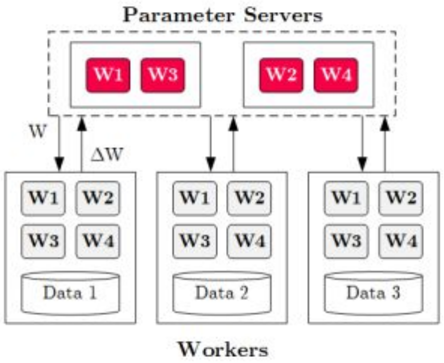
MAHOUT
APACHE Spark
Apache SystemML™

Execution Strategies

Mini-Batch Algorithms Smaller subset of data at a time -> improve computing time & memory usage

Parameter Server: centralizes the model parameters (e.g. NN weights) **->** multiple nodes read and update them.

- Data-parallel and model-parallel PS
- Update strategies (e.g., async, sync, backup)
- Data partitioning strategies (simple, featured-based)
- Federated ML
  - **Data** stays on **local devices**.
  - **Models** are **trained locally**, and only the **updated parameters** are sent to a **central server**.

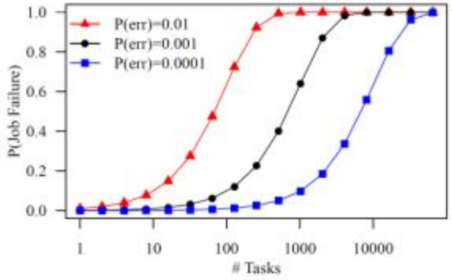


Execution Strategies

Lots of PS Decisions -> Acc/Perf-Tradeoff

- Configurations
  - Number of worker nodes
  - Batch size
  - Update strategie and frequency

Fault Tolerance & Resilience



Resilience Problem

- Increasing error rates at scale (soft/hard mem/disk/net errors)
- Robustness for interruptions

Fault Tolerance & 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

Language Abstractions

Apps
Lang
Faults
Exec
Data
HW

Optimization Scope

- Eager Interpretation (**no optimization**)
- Lazy expression evaluation (**some optimizations**)
- Program compilation (**full optimization**, difficult)

Optimization Objective

- Most common: **minimize time** under memory constraints.
- Multi-objective: **min cost** under time constraints, min time under accuracy constraints, **max accuracy** under time constraints

Language Abstractions

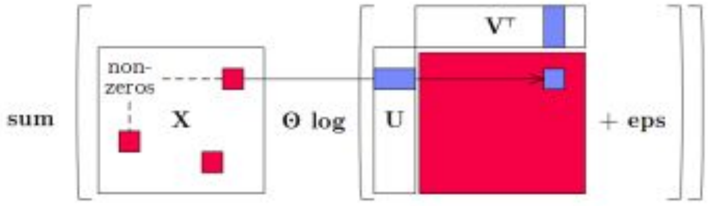
Apps
Lang
Faults
Exec
Data
HW

Trend: Fusion and Code Generation

- Custom fused operations

- Examples: SystemDS, Weld, Taco, T

Sparsity-Exploiting Operator



ML Applications

ML Algorithms (cost/benefit - time vs acc)

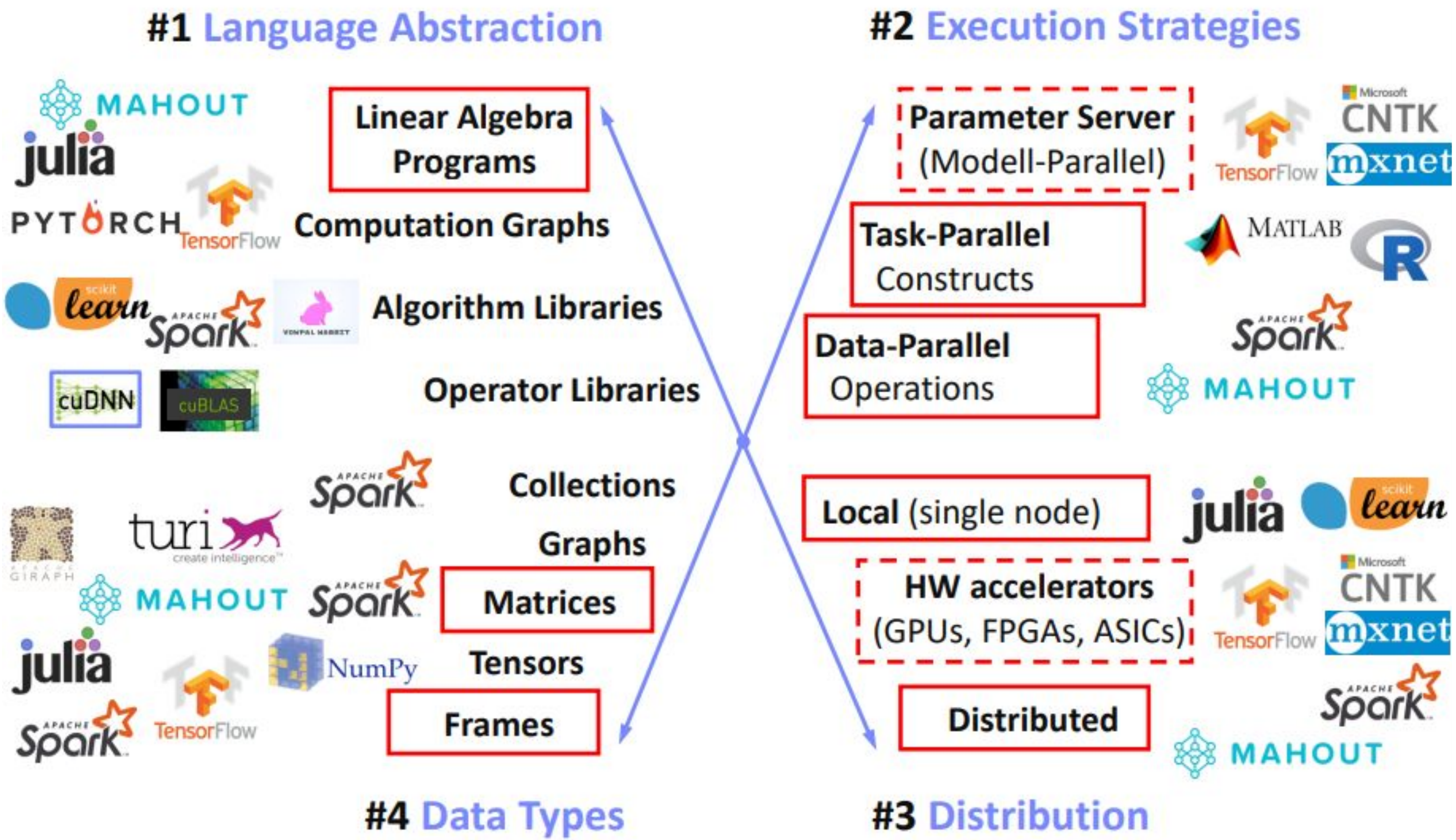
- Unsupervised/supervised; batch/mini-batch; first/second-order ML
- Mini-batch DL: variety of NN architectures and SGD optimizers

Specialized Apps: Video Analytics in NoScope (time vs accuracy)

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

[Credit: Daniel Kang‘17]

Landscape of ML Systems

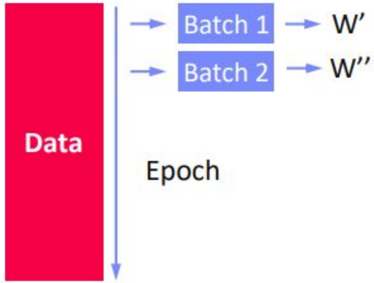


Distributed Parameter Servers

Background: Mini-batch ML Algorithms

Mini-batch ML Algorithms

- Iterative ML algorithms, where each iteration only uses a **batch of rows** to make the next model update
  - **Epochs** over the entire batch
  - **Random sampling** of the batch
- For **large** and highly redundant training sets
- Applies to **almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)



Background: Mini-batch ML Algorithms

Statistical vs Hardware Efficiency (batch size)

- **Statistical efficiency: more data** points to achieve **certain accuracy**
- **Hardware efficiency:** number of independent computations to achieve **high hardware utilization** (parallelization at different levels)
- **Batched recommended size:** 32 to 128 tuples

Background: Mini-batch DNN Training (LeNet)

[Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner: Gradient-Based Learning Applied to Document Recognition, **Proc of the IEEE 1998**]

```
# Initialize W1-W4, b1-b4
# Initialize SGD w/ Nesterov momentum optimizer
iters = ceil(N / batch_size)

for( e in 1:epochs ) {
   for( i in 1:iters ) {
      X_batch = X[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]
      y_batch = Y[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]

      ## layer 1: conv1 -> relu1 -> pool1
      ## layer 2: conv2 -> relu2 -> pool2
      ## layer 3:  affine3 -> relu3 -> dropout
      ## layer 4:  affine4 -> softmax
      outa4 = affine::forward(outd3, W4, b4)
      probs = softmax::forward(outa4)

      ## layer 4:  affine4 <- softmax
      douta4 = softmax::backward(dprobs, outa4)
      [doutd3, dW4, db4] = affine::backward(douta4, outr3, W4, b4)
      ## layer 3:  affine3 <- relu3 <- dropout
      ## layer 2: conv2 <- relu2 <- pool2
      ## layer 1: conv1 <- relu1 <- pool1

      # Optimize with SGD w/ Nesterov momentum W1-W4, b1-b4
      [W4, vW4] = sgd_nesterov::update(W4, dW4, lr, mu, vW4)
      [b4, vb4] = sgd_nesterov::update(b4, db4, lr, mu, vb4)
   }
}
```

NN Forward Pass

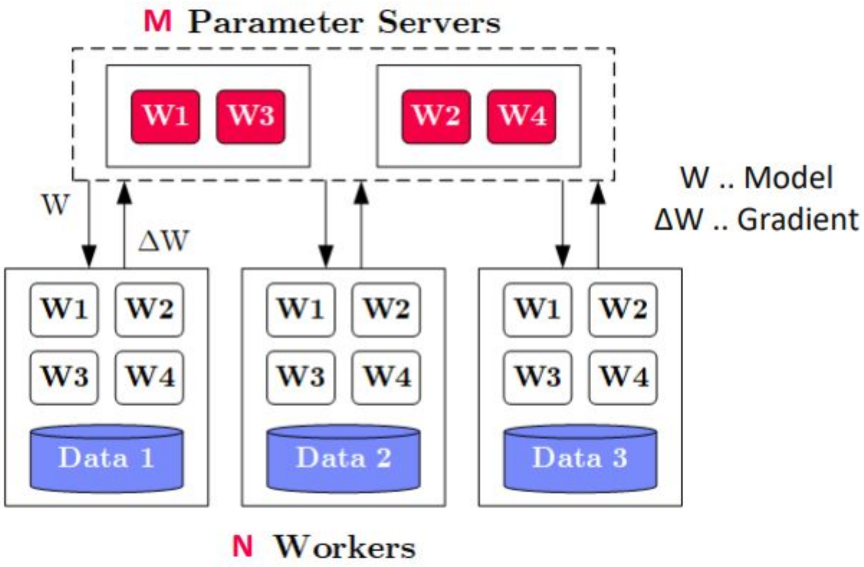
NN Backward Pass → Gradients

Model Updates

Overview Data-Parallel Parameter Servers

System Architecture

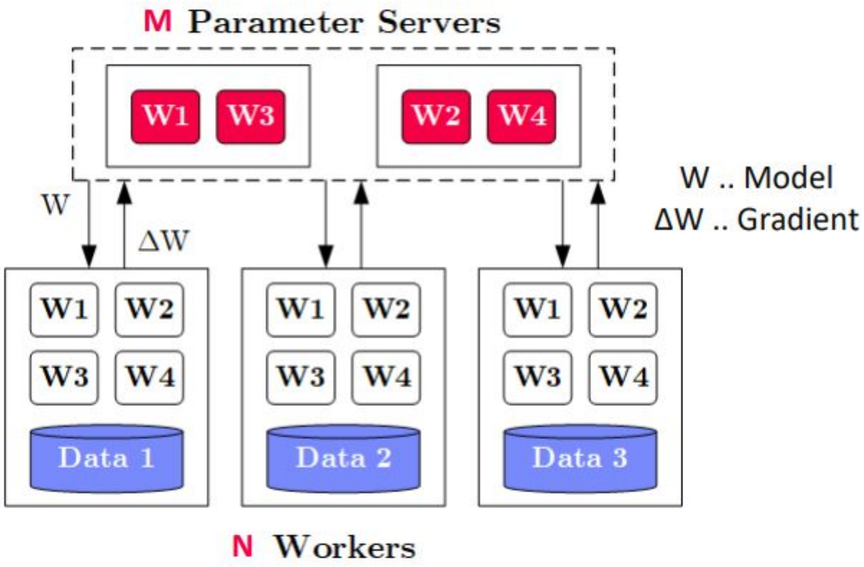
- **M:** Parameter Servers
- **N:** Workers
- Optimal Coordinator



Overview Data-Parallel Parameter Servers

System Architecture

- **M:** Parameter Servers
- **N:** Workers
- Optimal Coordinator



Key Techniques

- **Data partitioning D** -> workers Di (e.g., disjoint, reshuffling)
- Updated strategies (e.g., synchronous, asynchronous)
- Batch size strategies (small/large batches, hybrid methods)

History of Parameter Servers

1st Gen: Key/Value

- **Distributed key-value** store for parameter exchange and synchronization
- Relatively high overhead

2nd Gen: Classic Parameter Servers

- **Parameters as dense/sparse matrices**
- Different **update/consistency strategies**
- Flexible configuration and fault tolerance

[Alexander J. Smola, Shravan M. Narayanamurthy: An Architecture for Parallel Topic Models. **PVLDB 2010**]

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

[Mu Li et al: Scaling Distributed Machine Learning with the Parameter Server. **OSDI 2014**]

History of Parameter Servers

3rd Gen: Parameter Servers w/ improved data communication

- Prefetching and range-based pull/push
- Lossy or lossless compression w/ compensations

Examples

- TensorFlow, PyTorch

[Jiawei Jiang, Bin Cui, Ce Zhang, Lele Yu: Heterogeneity-aware Distributed Parameter Servers. **SIGMOD 2017**]

[Jiawei Jiang et al: SketchML: Accelerating Distributed Machine Learning with Data Sketches. **SIGMOD 2018**]

Basic Worker Algorithm (batch)

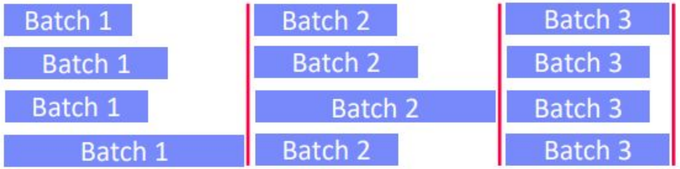
```
for( i in 1:epochs ) {
    for( j in 1:iterations ) {
        params = pullModel(); # W1-W4, b1-b4 lr, mu
        batch = getNextMiniBatch(data, j);
        gradient = computeGradient(batch, params);
        pushGradients(gradient);
    }
}
```

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Update Strategies

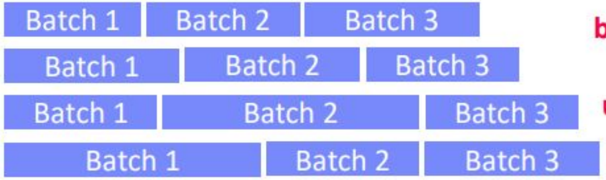
Bulk Synchronous Parallel (BSP)

- Update model w/ collected gradients
- Barrier for N workers



Asynchronous Parallel (ASP)

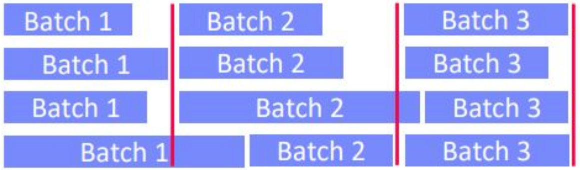
- Update model for each gradient
- No barrier



but, stale model updates

Synchronous w/ Backup Workers

- Update model w/collected gradients
- Barrier for N of N+b workers



[Martín Abadi et al: TensorFlow: A System for Large-Scale Machine Learning. **OSDI 2016**]

Intro to LLMs

Intro to LLMs

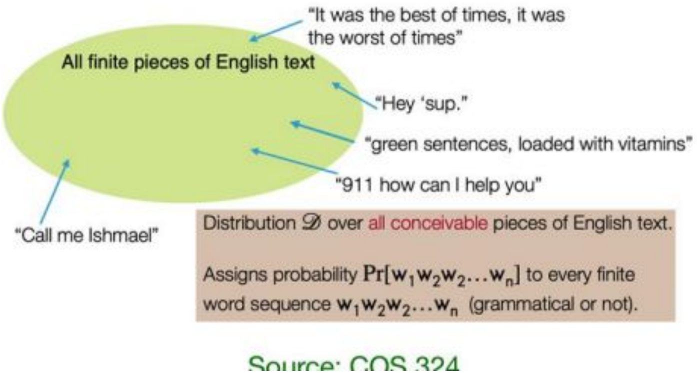
Options

- ChatGPT
- Google Gemini
- Lamda
- Llama
- Gork
- Mistral
- Eliza (1966)

	Gemini Ultra	Gemini Pro	GPT-4	GPT-3.5	PaLM 2-L	Claude 2	Inflection-2	Grok 1	LLAMA-2
MMLU Multiple-choice questions in 57 subjects (professional & academic) (Hendrycks et al., 2021a)	**90.04%** CoT@32* 83.7% 5-shot	79.13% CoT@8* 71.8% 5-shot	87.29% CoT@32 (via API**) 86.4% 5-shot (reported)	70% 5-shot	78.4% 5-shot	78.5% 5-shot CoT	79.6% 5-shot	73.0% 5-shot	68.0%***
GSM8K Grade-school math (Cobbe et al., 2021)	**94.4%** Maj1@32	86.5% Maj1@32	92.0% SFT & 5-shot CoT	57.1% 5-shot	80.0% 5-shot	88.0% 0-shot	81.4% 8-shot	62.9% 8-shot	56.8% 5-shot
MATH Math problems across 5 difficulty levels & 7 subdisciplines (Hendrycks et al., 2021b)	**53.2%** 4-shot	32.6% 4-shot	52.9% 4-shot (via API**) 50.3% (Zheng et al., 2023)	34.1% 4-shot (via API**)	34.4% 4-shot	—	34.8%	23.9% 4-shot	13.5% 4-shot
BIG-Bench-Hard Subset of hard BIG-bench tasks written as CoT problems (Srivastava et al., 2022)	**83.6%** 3-shot	75.0% 3-shot	83.1% 3-shot (via API**)	66.6% 3-shot (via API**)	77.7% 3-shot	—	—	—	51.2% 3-shot
HumanEval Python coding tasks (Chen et al., 2021)	**74.4%** 0-shot (IT)	67.7% 0-shot (IT)	67.0% 0-shot (reported)	48.1% 0-shot	—	70.0% 0-shot	44.5% 0-shot	63.2% 0-shot	29.9% 0-shot
Natural2Code Python code generation. (New held-out set with no leakage on web)	**74.9%** 0-shot	69.6% 0-shot	73.9% 0-shot (via API**)	62.3% 0-shot (via API**)	—	—	—	—	—
DROP Reading comprehension & arithmetic. (metric: F1-score) (Dua et al., 2019)	**82.4** Variable shots	74.1 Variable shots	80.9 3-shot (reported)	64.1 3-shot	82.0 Variable shots	—	—	—	—
HellaSwag (validation set) Common-sense multiple choice questions (Zellers et al., 2019)	87.8% 10-shot	84.7% 10-shot	**95.3%** 10-shot (reported)	85.5% 10-shot	86.8% 10-shot	—	89.0% 10-shot	—	80.0%***
WMT23 Machine translation (metric: BLEURT) (Tom et al., 2023)	**74.4** 1-shot (IT)	71.7 1-shot	73.8 1-shot (via API**)	—	72.7 1-shot	—	—	—	—

https://blog.google/technology/ai/google-gemini-ai/#sundar-note

Intro to LLMs



Options

- Next word prediction problem
- A probabilistic model that assign p**robability to every finite sequence** in e.g. **English language**
- Considering **context, position, grammar and structure**
- Sentence *"the cat sat on the mat"*

P(the cat sat on the map) = P(the)*P(cat|the) * P(sat|the cat) *P(on |the cat sat)*P(the|the cat sat on) *P(mat|the cat sat on the)

LLM Training

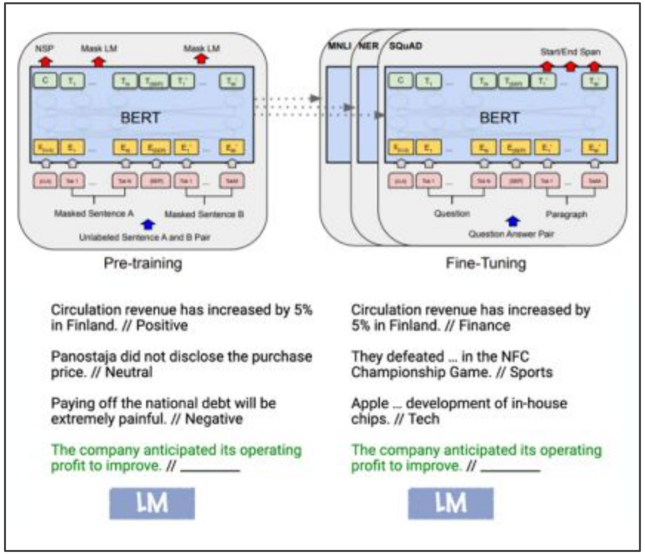
Transformer based neural networks

Pre-training (expensive)

- Download ~10TB of text.
- Get a cluster of ~6,000 GPUs.
- Compress the text into a neural
- network, pay ~$2M, wait ~12 days.
- Obtain base model.

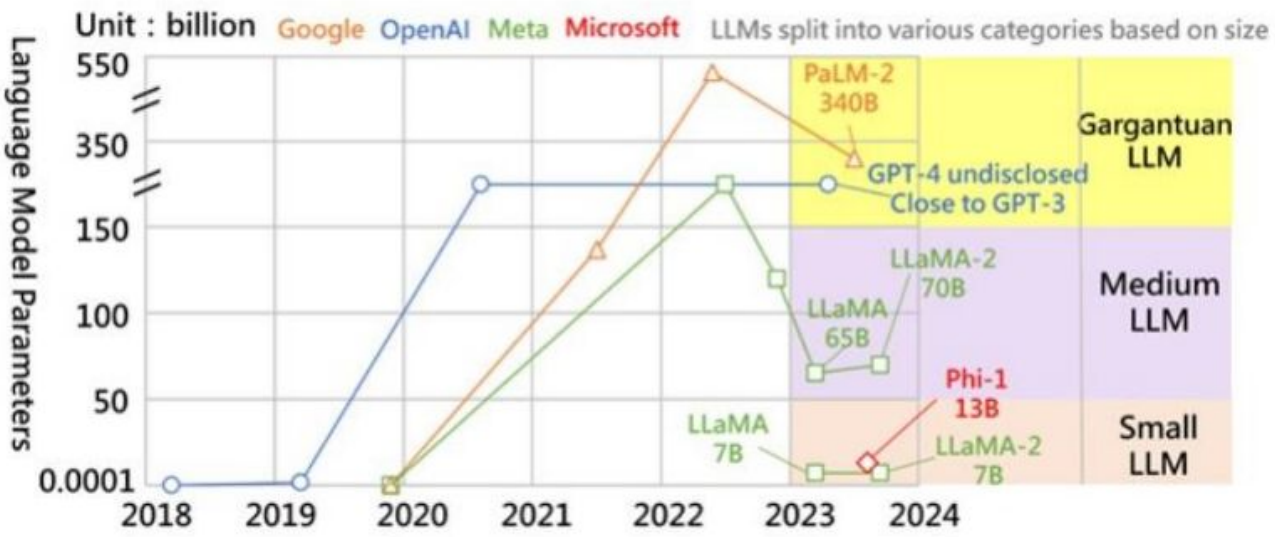
Fine Tuning

- Write labeling instructions
- Hire people (or use scale.ai!), collect 100K
- high quality ideal Q&A responses, and/or
- comparisons.
- Finetune base model on this data ~1 day.
- Obtain assistant model.
- Run a lot of evaluations.



LLM Parameters

Transformer based neural networks



Source: Compiled by DIGITIMES Research, Aug. 2023
https://www.digitimes.com/news/a20231221VL202/2024-outlook-ai-edge-ai-llm.html

Q & A and Exam Preparation

Multiple choice question

- **Which of the following best describes the concept of sparsification in Machine Learning?**
    a. Reducing the number of data points in a dataset by removing duplicates.
    b. Transforming a dense representation of data into a sparse one, where many values are zero.
    c. Increasing the density of data by adding synthetic samples to improve accuracy.
    d. Replacing categorical features with numerical representations for model compatibility.

Open questions

Message-oriented Middleware

- Describe the Message Delivery Guarantees At-Most-Once, At-Least-Once and Exactly-Once, and indicate which of them require persistent storage before sending.

Open questions

Message-oriented Middleware

- Describe the Message Delivery Guarantees At-Most-Once, At-Least-Once and Exactly-Once, and indicate which of them require persistent storage before sending.

Name	Description	Storage
At-Most-Once	Send and forget, never sent message twice (even on failures)	No
At-Least-Once	Store and forward, replay stream from (acknowledged) checkpoint	Yes
Exactly-Once	Store and forward, replay stream from (acknowledged) checkpoint, transactional delivery	Yes

Open questions

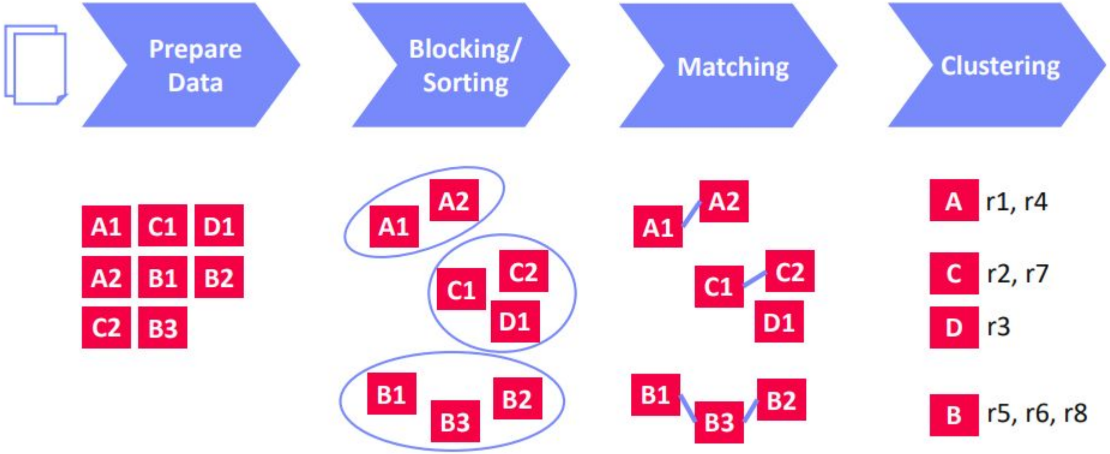
Schema Matching / Entity Linking

- Explain the phases of a typical Entity Resolution Pipeline with example techniques for the individual phases.

Open questions

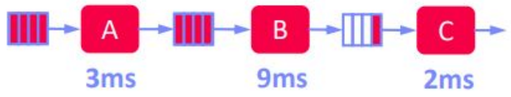
Schema Matching / Entity Linking

- Explain the phases of a typical Entity Resolution Pipeline with example techniques for the individual phases.



Stream Processing

a. **Back Pressure**
   - Graceful handling of overload w/o data loss
   - Slow down sources
   - E.g., blocking queues



Self-adjusting operator scheduling
Pipeline runs at rate of slowest op

b. **Load Shedding**
   - **Random-sampling-based** load shedding
   - **Relevance-based** load shedding
   - **Summary-based** load shedding (synopses)

Summary and Q&A

Summary and Q&A

- **Summary and Q&A**
  - Landscape of ML Systems
  - Distributed Parameter Servers
  - Large Language Models
  - Q&A and Exam Preparation
- **Oral Exam**
  - Starting **[Jan 30]**
- **Written Exam [Feb 07]**

Vielen Dank!

(please participate in the course evaluation)