Machine Learning Model Management and Inference (MLMMI)
02 Model Management Systems

Dr. Arnab Phani
BIFOLD, TU Berlin
DEEM Lab

22.04.2026

Advertisement: Thesis on WildMLBench

- **WildMLBench**
  - Olga Ovcharenko seeking a Bachelor/Master thesis student
  - **Benchmark for evaluating MLE agents** with realistic ML pipelines
  - **Non-contaminated multi-modal tasks**
  - Collect, create, curate datasets, implement ML pipelines, and evaluate MLE agents on the resulting benchmark.

Link

- **Required Qualification**
  - ML concepts, models, and workflows
  - Proficiency in ML libraries

- **Nice to Have**
  - Experience with MLE agents (e.g., AIDE)

Agenda

- **Model Management Systems**
- **Provenance for ML**
- **Feature Stores**
- **Modern Challenges**
- **Exercise 1: Model Management Systems**

Model Management Systems

Recap: Real-World ML

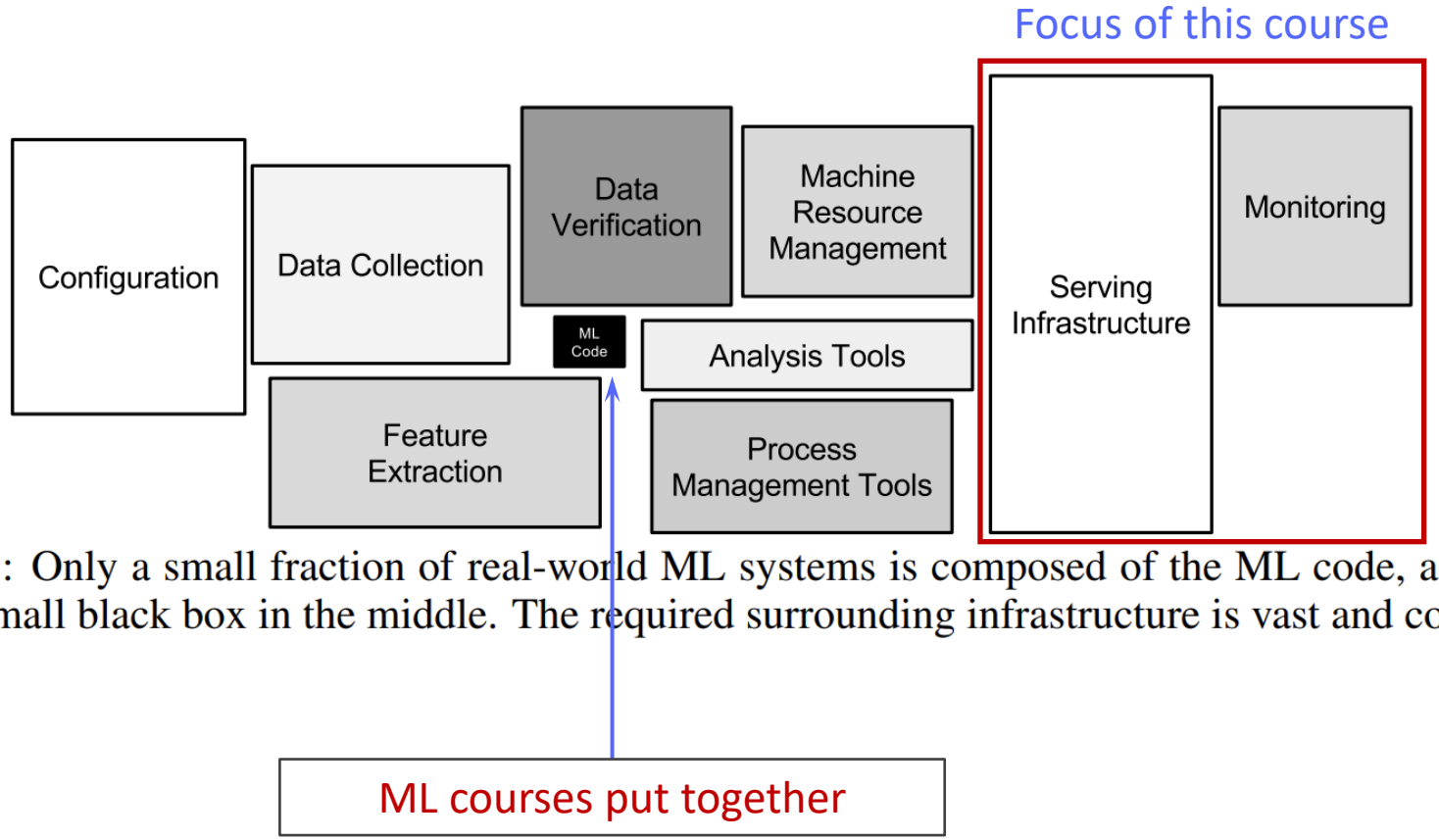


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Model Management: Overview

- **Motivation**
  - Data scientists build hundreds of models
  - Iterative and ad-hoc
  - Experiment + data + parameters
  - **New model built from previous models** and data
  - **Model reproducibility** (e.g., reverting to older model)
  - **Model sharing** in collaborative development
  - **Model governance** and explainability (e.g., GDPR)

Which model performed best on American customers? How did the two top models compare?

One ML developer spent over a week re-running a training another employee had previously conducted since the experimental setup and results were not recorded.

- **Challenges**
  - **Consistently** capture metadata: diversity of frameworks and libraries, connected via glue code
  - **Lack of declarative abstraction**
  - Metadata extraction by code instrumentation vs. external logging
  - Varying skill level

* MODELDB: Opportunities and Challenges in Managing Machine Learning Models. IEEE Data Engineering.

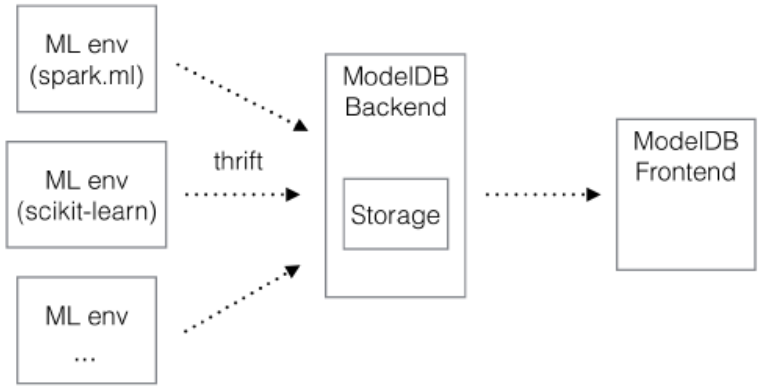
Example from Kaggle

```
# version 61
#   Drop fireplacecnt and fireplaceflag, following Jayaraman:
#     https://www.kaggle.com/valadi/xgb-w-o-outliers-lgb-with-outliers-combo-tune5

# version 60
#   Try BASELINE_PRED=0.0115, since that's the actual baseline from
#     https://www.kaggle.com/aharless/oleg-s-original-better-baseline

# version 59
#   Looks like 0.0056 is the optimum BASELINE_WEIGHT

# versions 57, 58
#   Playing with BASELINE_WEIGHT parameter:
#     3 values will determine quadratic approximation of optimum

...

# version 49
#   My latest quadratic approximation is concave, so I'm just taking
#     a shot in the dark with lgb_weight=.3

# version 45
#   Increase lgb_weight to 0.25 based on new quadratic approximation.
#   Based on scores for versions 41, 43, and 44, the optimum is 0.261
#     if I've done the calculations right.
#   I'm being conservative and only going 2/3 of the way there.
#   (FWIW my best guess is that even this will get a worse score,
#    but you gotta pay some attention to the math.)

# version 44
#   Increase lgb_weight to 0.23, per Nikunj's suggestion, even though
#     my quadratic approximation said I was already at the optimum
```

Top Kaggle competitors often submit hundreds of solutions

ModelDB

- Model and provenance logging for ML pipelines via programmatic APIs



Verta

IDs	DataFrame	Specifications	Metrics	Misc.
Model ID: 22 Experiment Run ID: 1 Experiment ID: 1	DataFrame ID: 30	Type: LinearRegression Hyperparameters	rmse: 0.881	Notes: test annotation Model Filepath: 2016-12-09 ... Timestamp: 2016-12-09 18:5... See Mo
Model ID: 29 Experiment Run ID: 1 Experiment ID: 1	DataFrame ID: 35	Type: LinearRegression Hyperparameters	rmse: 0.849	Notes: this model is funky Model Filepath: 2016-12-09 ... Timestamp: 2016-12-09 18:5... See Mo
Model ID: 30 Experiment Run ID: 1 Experiment ID: 1	DataFrame ID: 36	Type: LinearRegression Hyperparameters	rmse: 0.873	Notes: feature1, feature2, fe... Model Filepath: 2016-12-09 ... Timestamp: 2016-12-09 18:5... See Mo
Model ID: 31 Experiment Run ID: 1 Experiment ID: 1	DataFrame ID: 37	Type: LinearRegression Hyperparameters	rmse: 0.942	Notes: Model Filepath: 2016-12-09 ... Timestamp: 2016-12-09 18:5... See Mo

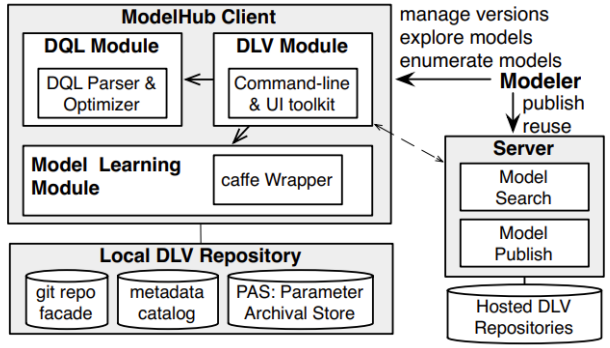
```
run = client.set_experiment_run("First Run")
run.log_hyperparameters({"regularization": 0.5})
# ... model training code goes here
run.log_metric('accuracy', 0.72)

run = client.set_experiment_run("Second Run")
run.log_hyperparameters({"regularization": 0.8})
# ... model training code goes here
run.log_metric('accuracy', 0.83)
```

MODELDB tracks models as they are built, records provenance information for each step in the pipeline, stores this data in a standard format (PostgreSQL), and makes it available for querying via an API and a GUI.

* MODELDB: Opportunities and Challenges in Managing Machine Learning Models. IEEE Data Engineering.

ModelHub

- Model versioning system for DNNs
- Provenance tracking
- DSL for model exploration and enumeration queries
- Model metadata stored as deltas



Type	Command	Description
model version management	`init`	Initialize a `dlv` repository.
	`add`	Add model files to be committed.
	`commit`	Commit the added files.
	`copy`	Scaffold model from an old one.
	`archive`	Archive models in the repository.
model exploration	`list`	List models and related lineages.
	`desc`	Describe a particular model.
	`diff`	Compare multiple models.
	`eval`	Evaluate a model with given data.
model enumeration	`query`	Run DQL clause.
remote interaction	`publish`	Publish a model to ModelHub.
	`search`	Search models in ModelHub.
	`pull`	Download from ModelHub.

Model versioning utilities

```
select m1
where  m1.name like "alexnet_%" and
       m1.creation_time > "2015-11-22" and
       m1["conv[1,3,5]"].next has POOL("MAX")
```

Query 1. DQL `select` query to pick the models.

```
slice m2 from m1
where m1.name like "alexnet-origin%"
mutate m2.input = m1["conv1"] and
       m2.output = m1["fc7"]
```

Query 2. DQL `slice` query to get a sub-network.

```
construct m2 from m1
where  m1.name like "alexnet-avgv1%" and
       m1["conv*($1)"].next has POOL("AVG")
mutate m1["conv*($1)"].insert = RELU("relu$1")
```

Query 3. DQL `construct` query to derive more models on existing ones.

```
evaluate m
from "query3"
with config = "path to config"
vary config.base_lr in [0.1, 0.01, 0.001] and
     config.net["conv*"].lr auto and
     config.input_data in ["path1", "path2"]
keep top(5, m["loss"], 100)
```

Query 4. DQL `evaluate` query to enumerate models with different network architectures, search hyperparameters, and eliminate models.

* ModelHub: Deep Learning Lifecycle Management. ICDE 2017

DataHub and ProvDB

- **DataHub**
  - Heteregenous datasets; combine relational and unstructured
  - Copy and modify is expensive (data duplication)
  - Capture **dataset versions**
  - Commands: create, branch, merge, commit, rollback, checkout
  - One materialized version with other **non-materialized deltas**
  - Significantly save storage
  - **Storage vs. efficiency**
  - Similar to **MVCC** in database systems

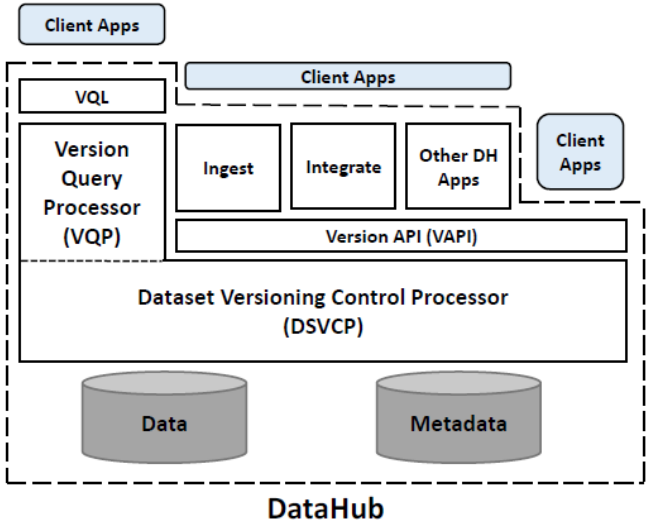


Figure 1: DataHub Components and Architecture.

- **ProvDB**
  - Hard to define a schema for provenance/metadata
  - Schema-later: Fixed base schema, arbitrary semi-structured JSON

Q: Which derived datasets need to be updated when a source dataset changes?

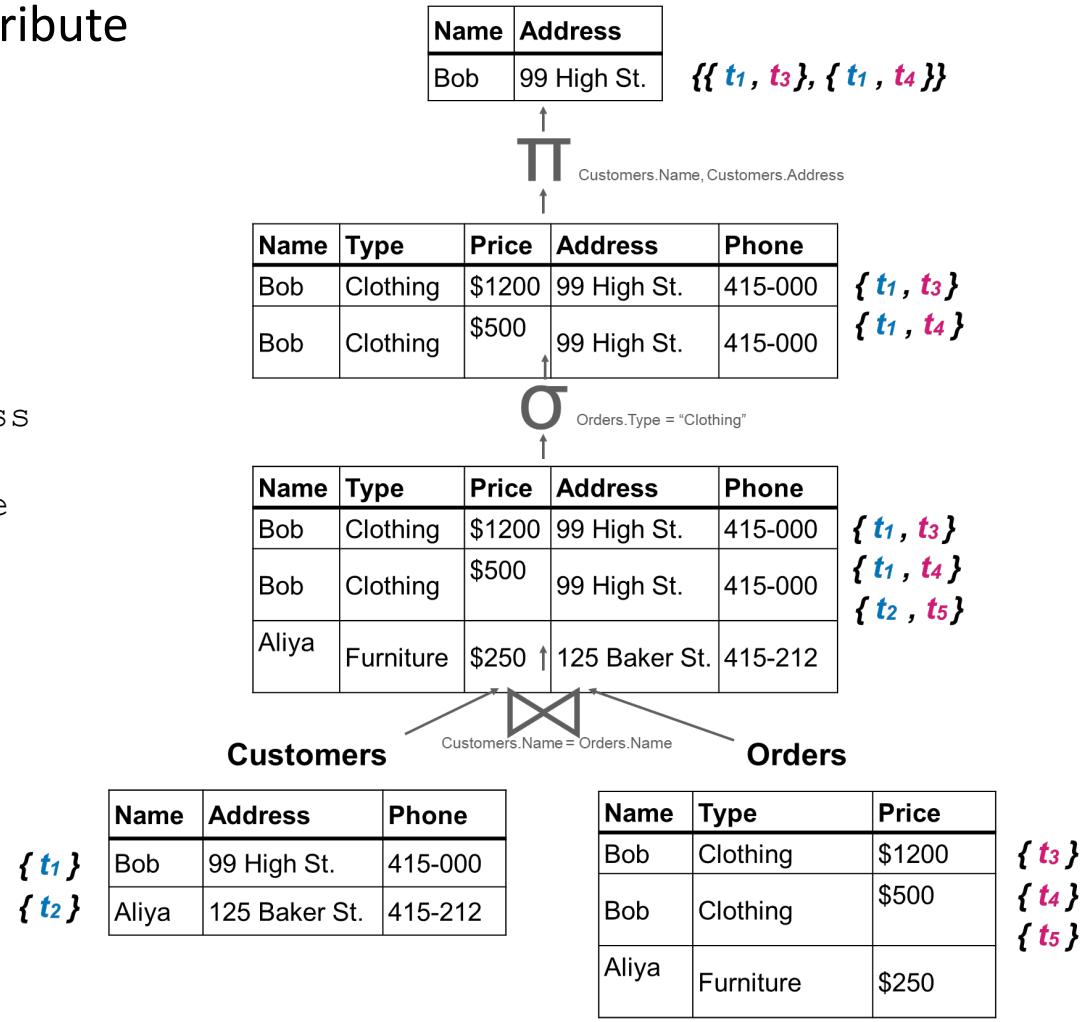
* DataHub: Collaborative Data Science & Dataset Version Management at Scale. CIDR 2015.
* ProvDB: Lifecycle Management of Collaborative Analysis Workflows. HILDA@SIGMOD 2027

Provenance for ML

Bonus: Data Provenance

- **Why-Provenance:** set of all input tuples that contribute to the result (referred to as “witnesses”)

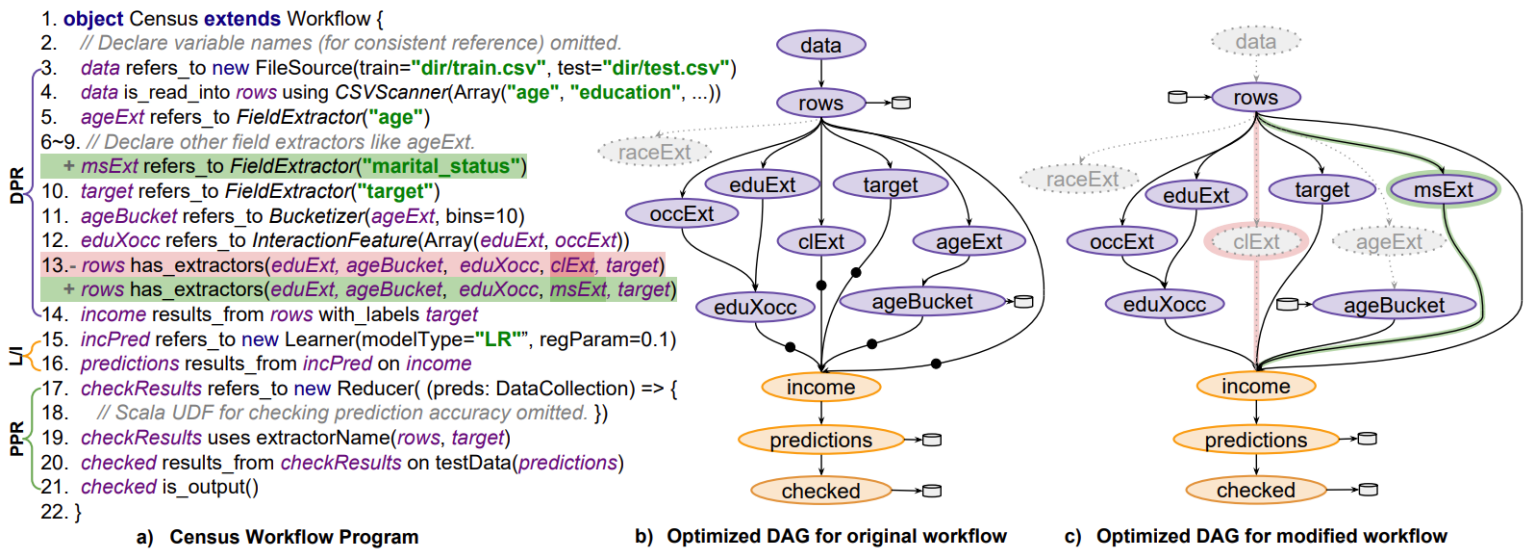
```
SELECT DISTINCT Customers.Name, Customers.Address
    FROM Customers
      JOIN Orders ON Customers.Name = Orders.Name
WHERE Orders.Type = “Clothing”
```



Provenance for ML Pipelines

- **Helix**
  - **Iterative development** with small modifications
  - Coarse-grained lineage tracing, caching, reuse
  - Cost-based optimization
  - Materialization to disk for future reuse

Which intermediates to cache? – NP-HARD
Which intermediates to reuse? – MAX-FLOW-based algorithm



a) Census Workflow Program b) Optimized DAG for original workflow c) Optimized DAG for modified workflow

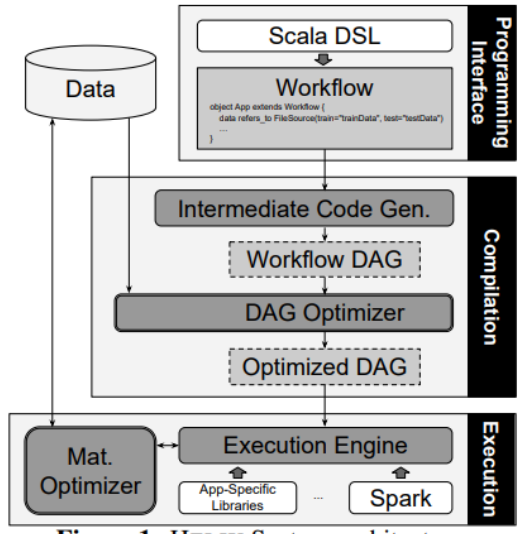
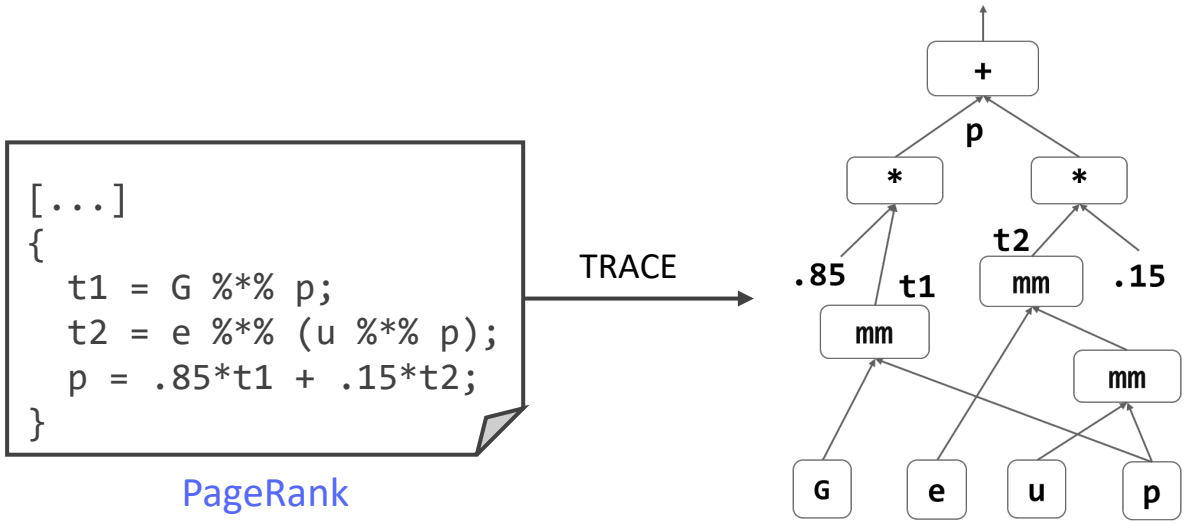


Figure 1: HELIX System architecture.

* HELIX: Holistic Optimization for Accelerating Iterative Machine Learning. PVLDB 2018.

LIMA: Fine-grained Lineage Tracing

LIMA



- **Lineage DAG**
  - Trace lineage and use lineage traces for **reuse, explainability, debugging, and re-computation**
  - During **runtime of a linear algebra program**, we maintain lineage DAGs for all live variables
  - Incrementally built as we execute instructions

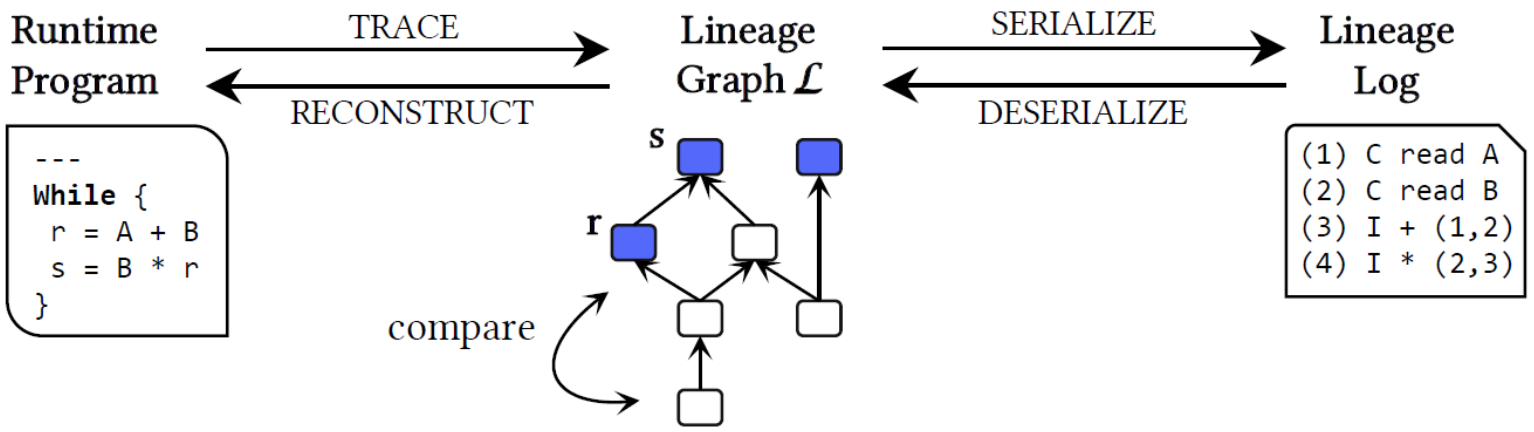
* LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. SIGMOD 2021.

Apache SystemDS ™

LIMA: Key Operations

LIMA

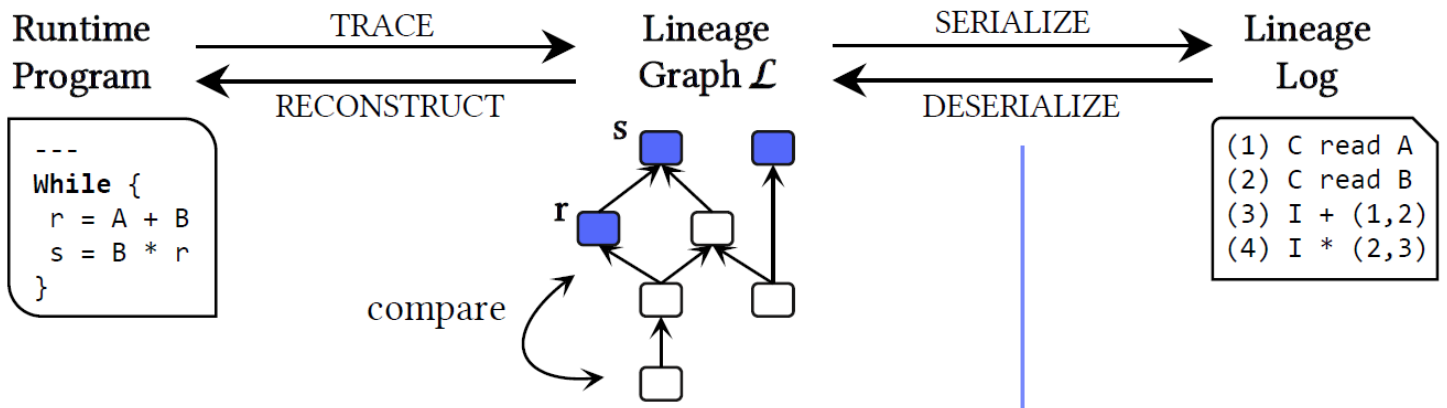
Lineage Tracing Lifecycle



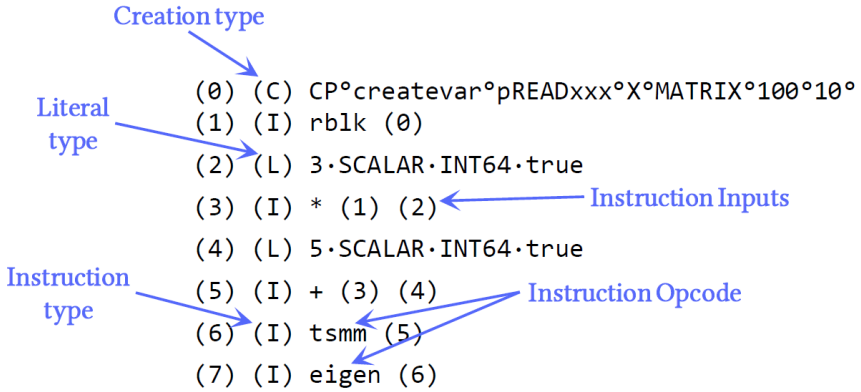
The entire lifecycle of lineage tracing with the key operations is very valuable as it simplifies **testing, debugging, reproducibility, and reuse**.

Key Operations

LIMA



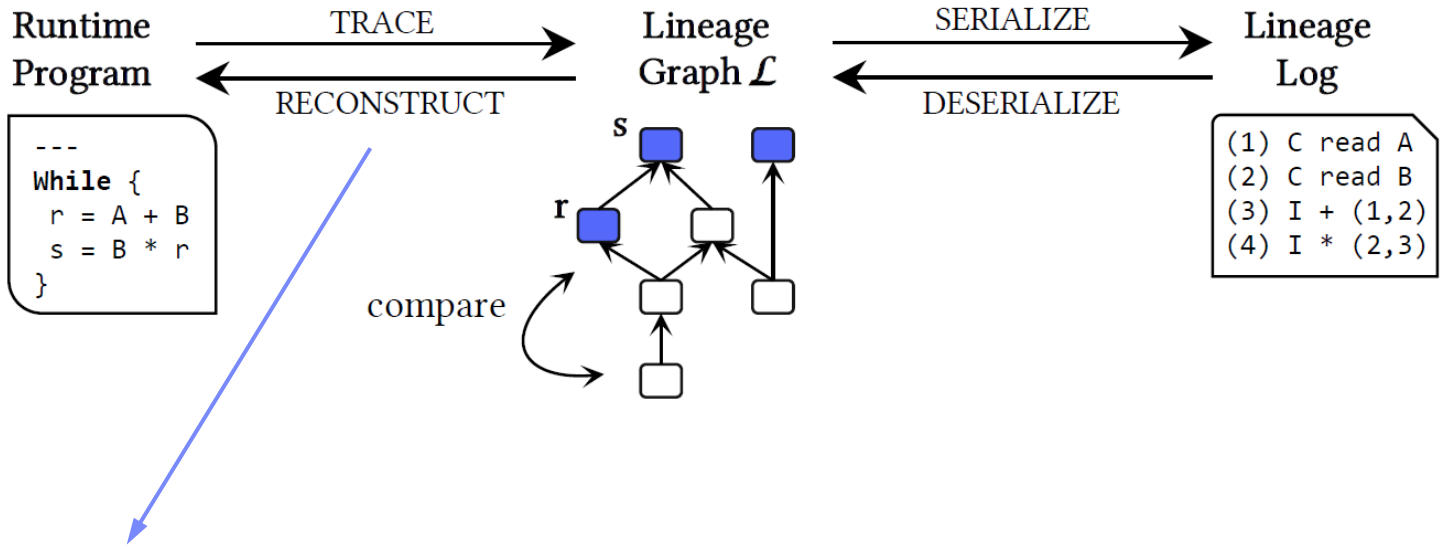
Lineage Tracing Lifecycle



- **Serialization and Deserialization of Lineage DAGs**
  - `lineage(X)`, and `write(X, 'f')` always generates `'f.lineage'`
  - Serialization unrolls the DAG in a depth-first manner
  - Deserialization converts lineage log into a lineage DAG

Key Operations, cont.

LIMA

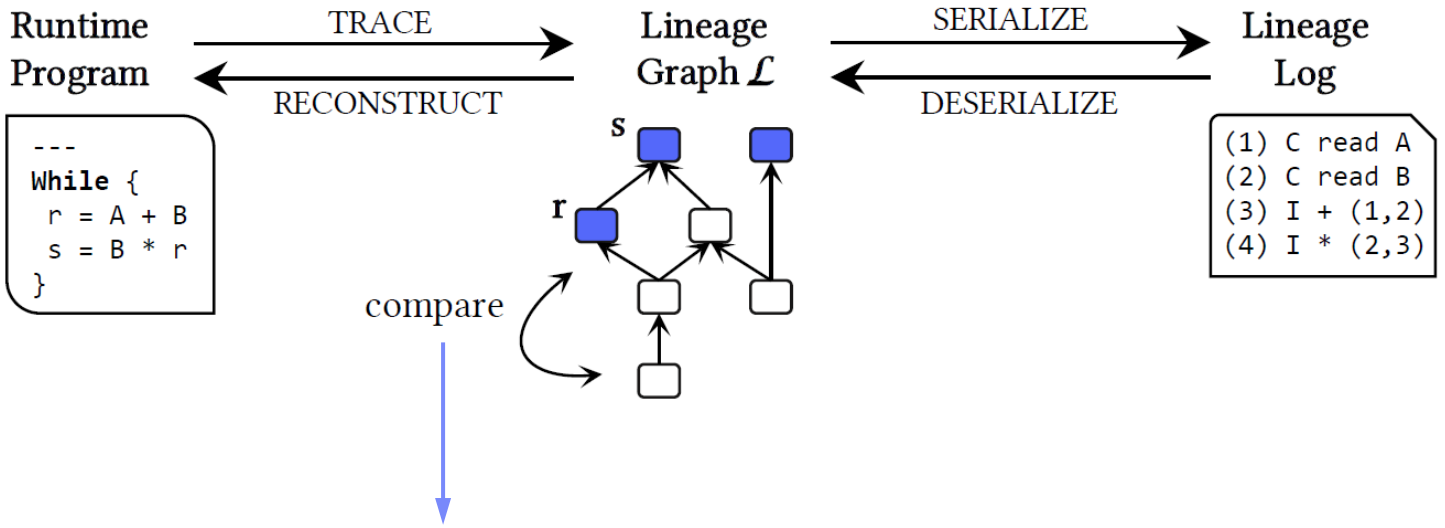


Lineage Tracing Lifecycle

- **Re-computation from Lineage**
  - Generate runtime program from a lineage DAG
  - **Compute exactly the same intermediates**
  - Does not contain control flow

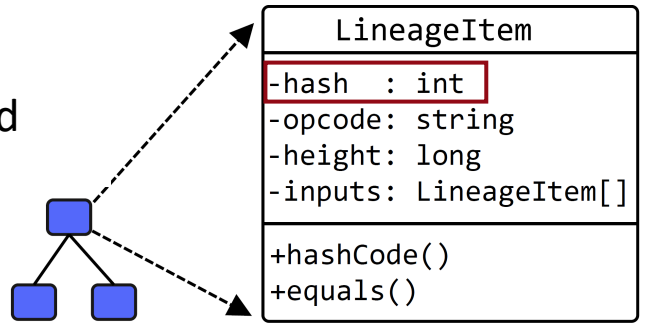
Key Operations, cont.

LIMA



Lineage Tracing Lifecycle

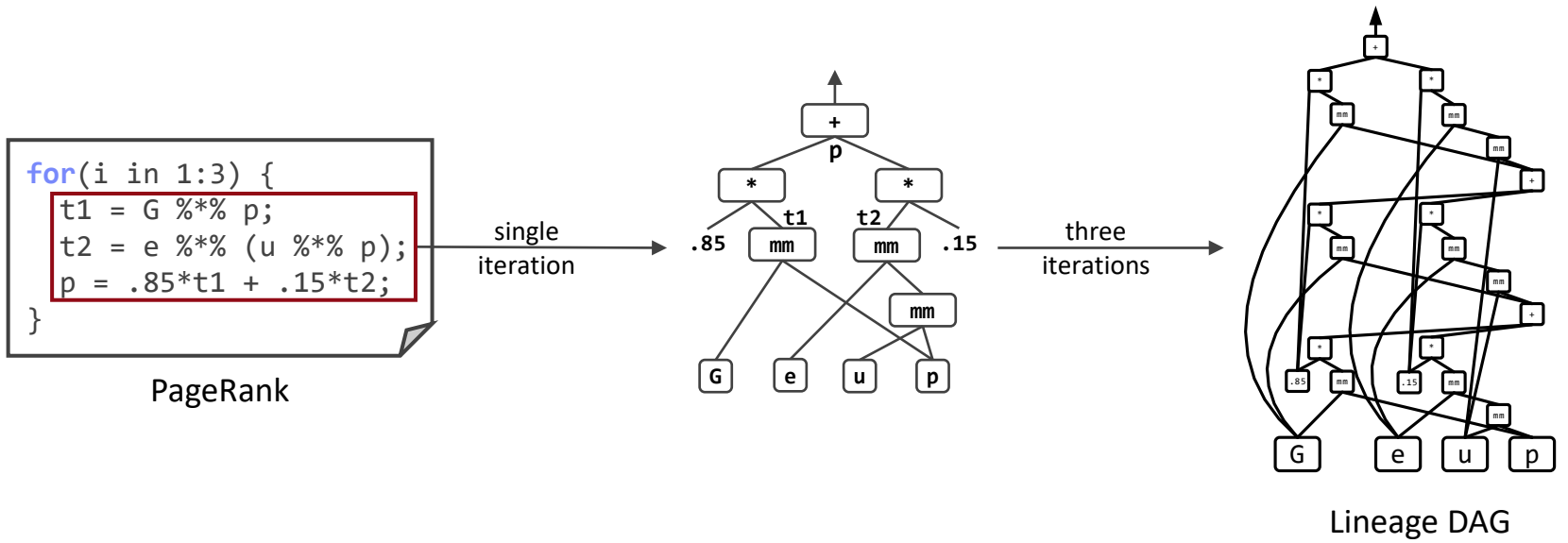
- **Comparison of Lineage DAGs**
  - Lineage items implement **`hashCode()`** and **`equals()`**
  - Hash over the hashes of opcode, data item, and all inputs
  - **Non-recursive** equals; returns true if the opcode, data and all inputs are equivalent



Lineage Deduplication

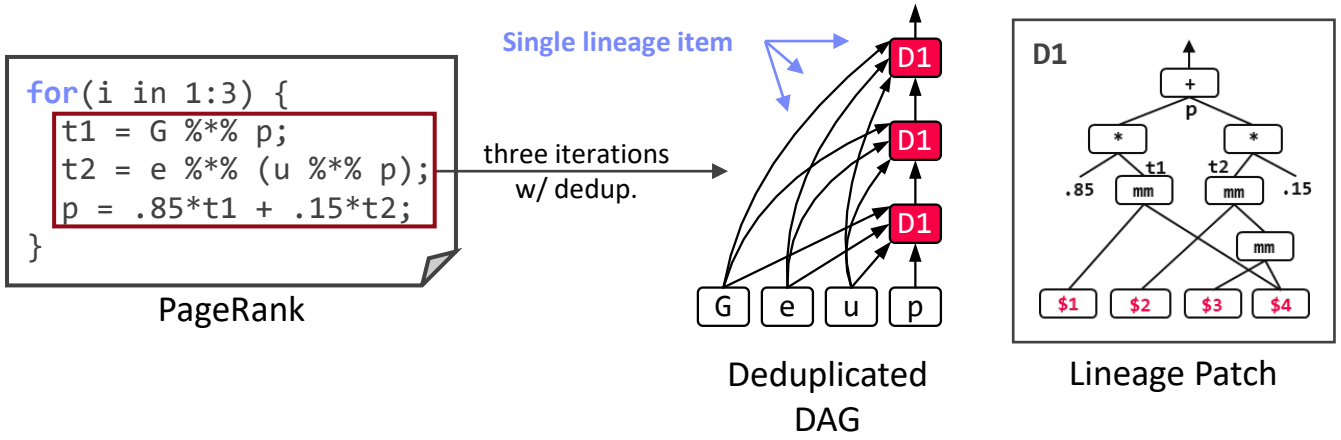
LIMA

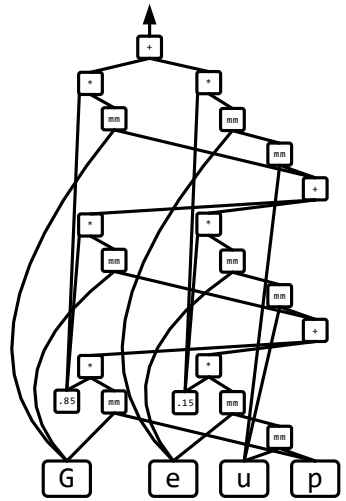
- **Problem**
  - Very **large lineage DAGs** for mini-batch training (repeated execution of loop bodies)
  - NN training w/ 200 epochs, batch-size 32, 10M rows, 1K instructions → 4TB → **4GB w/ deduplication**



Lineage Deduplication, cont.

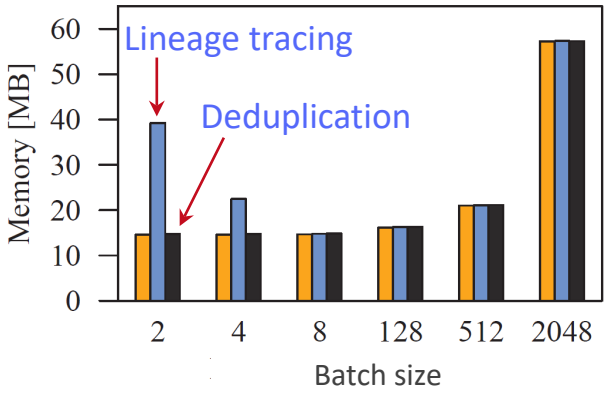
LIMA





Lineage DAG
w/o Deduplication

- **Solution**
  - Trace **each independent path once**; store as patches
  - Refer to the patches via **single lineage items**

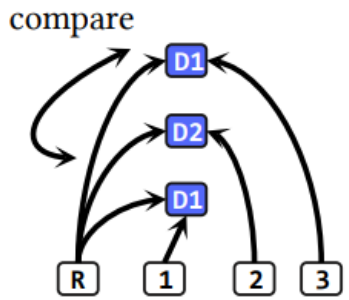


Reproducibility and Debuggability

Program

```
R = rand(10, 5)
for (i in 1:3) {
 R = R + 1/2;
 if (i %% 2 == 0)
  R = R * 3*i;
 R = R - 5;
}
R = R * 4;
```

Lineage Graph



Lineage Log

```
(1) C rand R
(2) D dedup1 (1,'1')
(3) D dedup2 (1,'2')
(4) D dedup1 (1,'3')
(5) I * (4,'4')
D1:
(1) I + ($1,'0.5')
(2) I - (1,'5')
D2:
(1) I + ($1,'0.5')
(2) I * (1,'3')
(3) I * (2,$2)
(4) I - (3,'5')
```

Recomputed Prog.

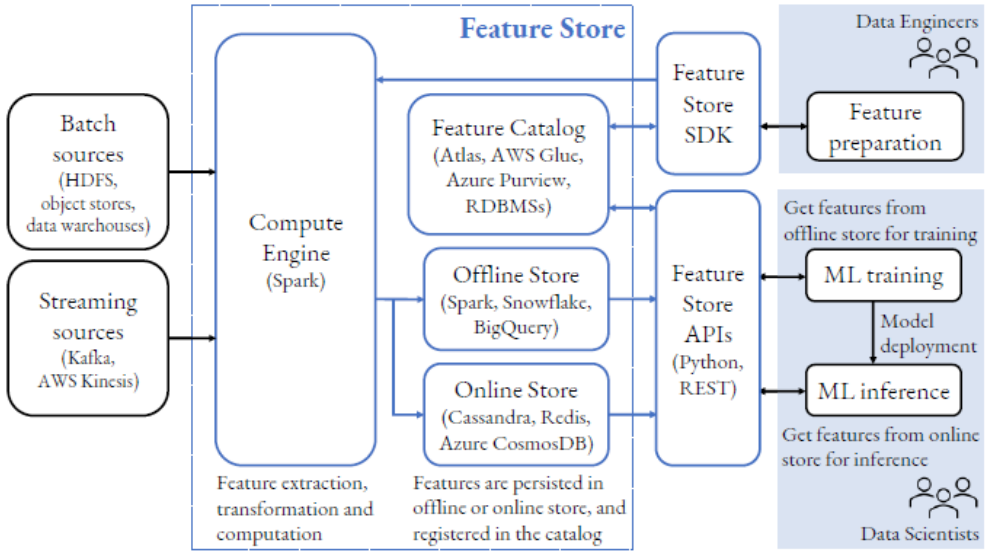
```
Dedup1 = function(IN0,IN1){
 tmp1 = IN0 + 0.5
 tmp2 = tmp1 - 5
 return tmp2 }
Dedup2 = function(IN0,IN1){
 tmp1 = IN0 + 0.5
 tmp2 = tmp1 * 3
 tmp3 = tmp2 * IN1
 tmp4 = tmp3 - 5
 return tmp4 }
R = rand(10, 5)
R = Dedup1(R, 1)
R = Dedup2(R, 2)
R = Dedup1(R, 3)
R = R * 4
```

- **Re-computation from Lineage**
  - Reconstruct runtime program from lineage log
  - **Produce same output for same inputs**
- **Debugging ML Pipelines**
  - Future Research: Query processing over lineage traces

A small change in an ML script may have significant impact on the final result, but identifying that change may be non-trivial, especially in a collaborative setting;

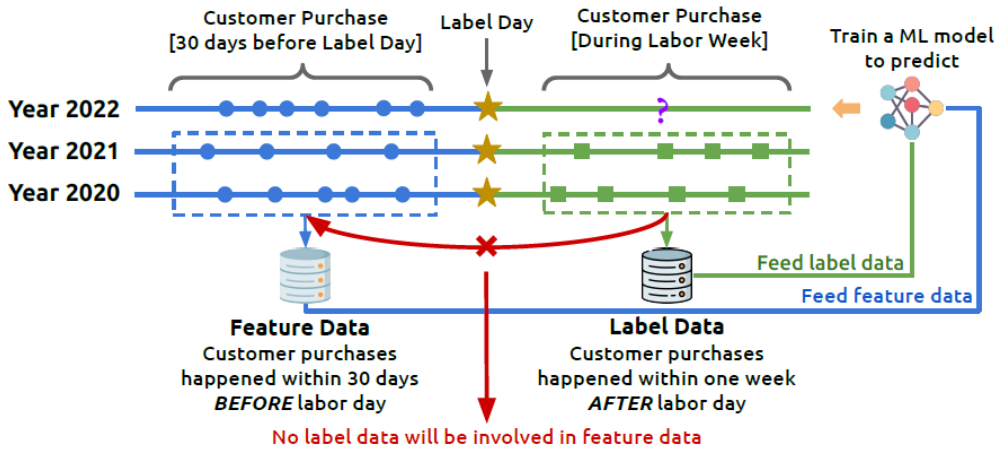
Feature Stores

Feature Stores



- **Overview**
  - **Central repository for features.** Allow users to create complex pipelines, composed of multiple stages, for transforming raw data into relevant features.
  - Handels primarily **time-series data**
  - **APIs** for operations and point-in-time join
  - Features written to **offline stores** (training, high throughput) and **online stores** (inference, low-latency)

Point-In-Time Correctness



***Label Source* Dataset**

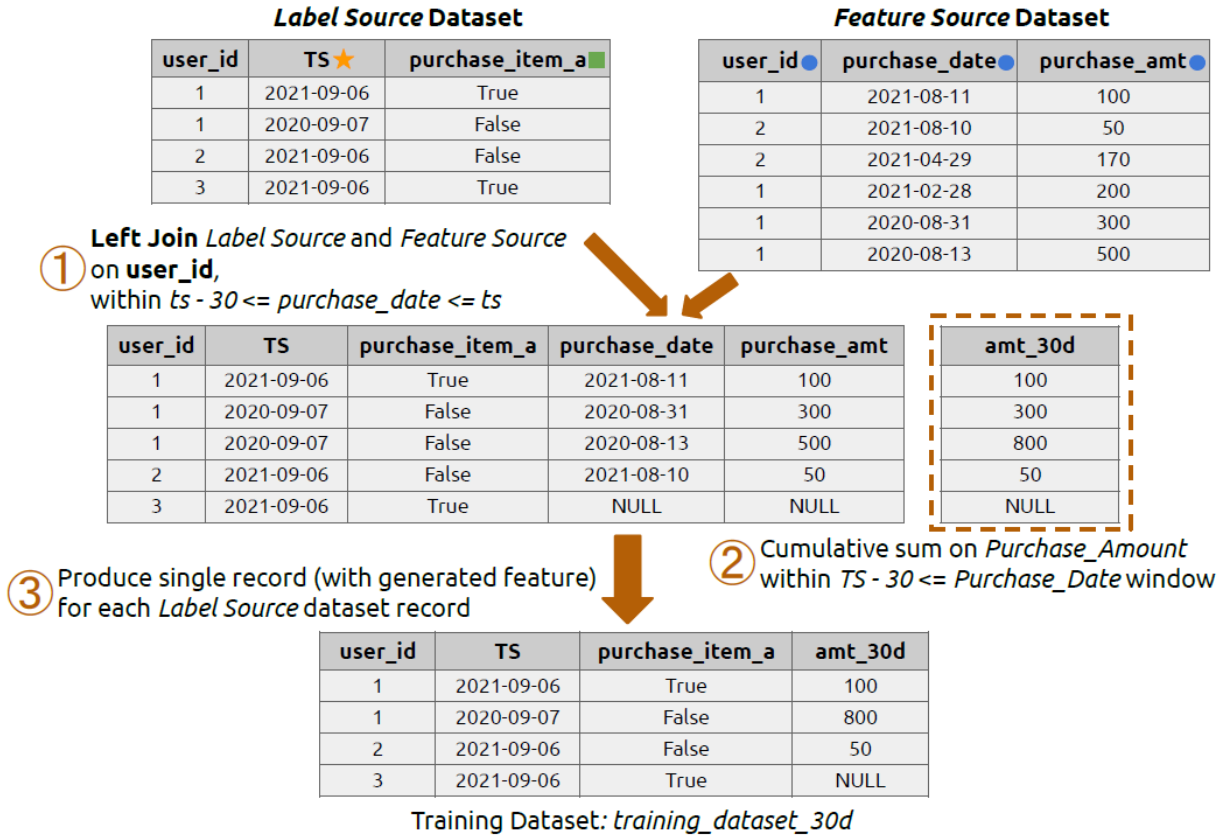
user_id	TS	purchase_item_a
1	2021-09-06	True
1	2020-09-07	False
2	2021-09-06	False
3	2021-09-06	True

***Feature Source* Dataset**

user_id	purchase_date	purchase_amt
1	2021-08-11	100
2	2021-08-10	50
2	2021-04-29	170
1	2021-02-28	200
1	2020-08-31	300
1	2020-08-13	500

- **Point-In-Time Join**
  - **Core operation** for generating training dataset
  - Sources combined according to specific point-in-time (cutoff point); avoid data leakage; window agg
  - Example: whether a customer will buy item `a` during Labor Week of 2022

Point-In-Time Correction Example




```sql
SELECT user_id, ts, purchase_item_a, amt_30d
FROM label_src_dataset l_ds
  LEFT JOIN LATERAL (
    SELECT SUM (purchase_amt)
        OVER (ORDER BY purchase_date ASC) AS amt_30d
    FROM feature_src_dataset f_ds
    WHERE f_ds.user_id = l_ds.user_id
      AND f_ds.purchase_date >= l_ds.ts - 30
      AND f_ds.purchase_date <= l_ds.ts
    ORDER BY purchase_date DESC LIMIT 1) subq ON TRUE;
```

(a) PIT join workflow q_1 with 30 days window aggregate used to compute `training_dataset_30d`.

```python
# define PIT join key
join_id = TypedKey(key_column="user_id", key_type="ValueType.INT32")
# define window aggregated feature
agg = WindowAggTransformation(expr="purchase_amt", agg_func="SUM", window="30d")
feature = Feature(name="amt_30d", key=join_id, transform=agg)
# define label source dataset
lsd = HDFSSource(data_path=label_src_dataset, timestamp_column="ts")
# define feature source dataset
fsd = HDFSSource(data_path=feature_src_dataset, timestamp_column="purchase_date")
# define feature based on PIT join
feature = FeatureAnchor("training_dataset_30d", [lsd, fsd], [feature])
```

* Optimizing Data Pipelines for Machine Learning in Feature Stores. PVLDB 2023.

FeathrPO: Reuse-Based Feature Computation

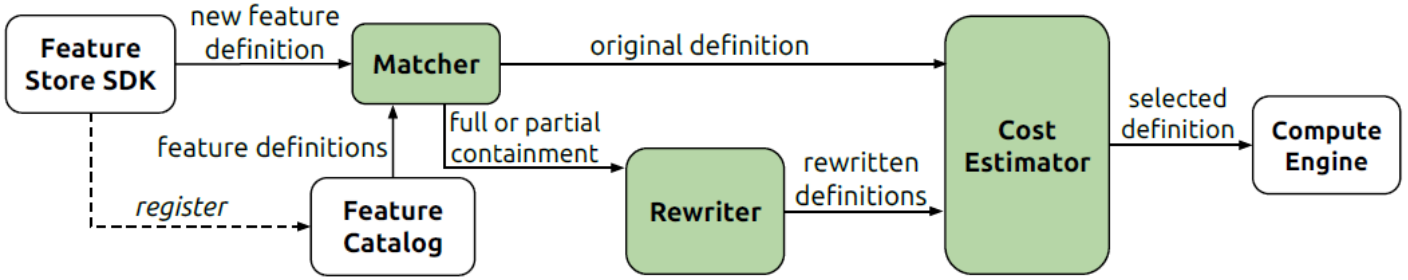


Figure 5: Feature extraction optimization workflow.

- **Overview**
  - Rewrite incoming pipelines to **reuse previously computed features**
  - Exact and partial match (e.g., time window overlap between features)
  - Rewrite the query to compute join over delta

* Optimizing Data Pipelines for Machine Learning in Feature Stores. PVLDB 2023.

FeathrPO: Reuse-Based Feature Computation, cont.

```
SELECT user_id, ts, purchase_item_a,
  CASE
    WHEN amt_30d IS NULL AND amt IS NULL THEN NULL
    ELSE COALESCE(amt_30d, 0) + COALESCE(amt, 0)
  END AS amt_40d
FROM (
  SELECT user_id, ts, purchase_item_a, amt_30d, amt
  FROM training_dataset_30d t_ds
    LEFT JOIN LATERAL (
      SELECT SUM (purchase_amt)
          OVER (ORDER BY purchase_date ASC) AS amt
      FROM feature_src_dataset f_ds
      WHERE f_ds.user_id = t_ds.user_id
        AND f_ds.purchase_date >= t_ds.ts - 40
        AND f_ds.purchase_date < t_ds.ts - 30
      ORDER BY purchase_date DESC LIMIT 1) subq ON TRUE
) o_subq;
```

(c) Reuse-based optimized workflow q'_2 using `training_dataset_30d` computed by q_1.

```
WITH agg_t_ds AS (
  SELECT MIN (ts) AS min_ts, MAX (ts) AS max_ts
  FROM training_dataset_30d
)
SELECT user_id, ts, purchase_item_a,
  CASE
    WHEN amt_30d IS NULL AND amt IS NULL THEN NULL
    ELSE COALESCE(amt_30d, 0) + COALESCE(amt, 0)
  END AS amt_40d
FROM (
  SELECT user_id, ts, purchase_item_a, amt_30d, amt
  FROM training_dataset_30d t_ds
    LEFT JOIN LATERAL (
      SELECT SUM (purchase_amt)
          OVER (ORDER BY purchase_date ASC) AS amt
      FROM (
        SELECT f_ds.* FROM feature_src_dataset f_ds, agg_t_ds
        WHERE f_ds.purchase_date >= agg_t_ds.min_ts - 40
          AND f_ds.purchase_date < agg_t_ds.max_ts - 30
      ) f_ds
      WHERE f_ds.user_id = t_ds.user_id
        AND f_ds.purchase_date >= t_ds.ts - 40
        AND f_ds.purchase_date < t_ds.ts - 30
      ORDER BY purchase_date DESC LIMIT 1) subq ON TRUE
) o_subq;
```

(d) Optimized workflow q''_2 after introducing semijoin reduction on q'_2.

- **Reuse in PIT Join Pipeline**
  - The filter in the inner subquery is altered to only **compute the join over the remaining delta**
  - Reduce the amount of data processed in the inner query via **semi-join reduction**.
  - Cost-based rewrite: $C_q = \sum_{s \in S_q} \mathcal{D}_s - \mathcal{U}^q_{s_p}$

FeathrPO: Automatic Layout Selection

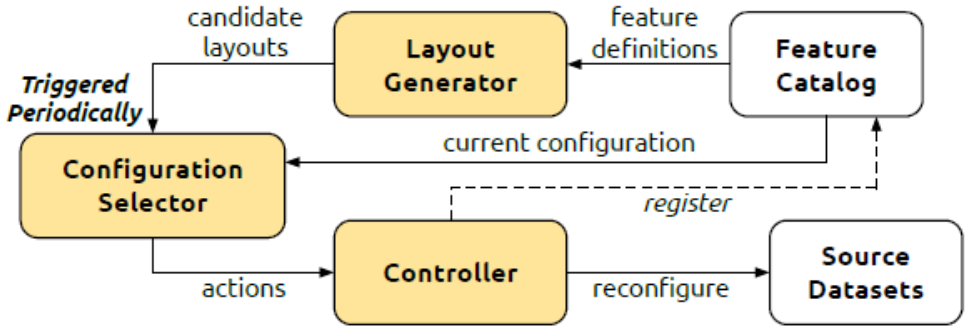
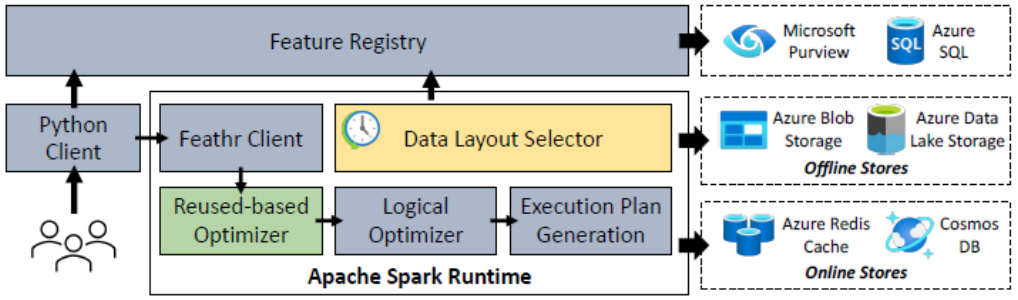


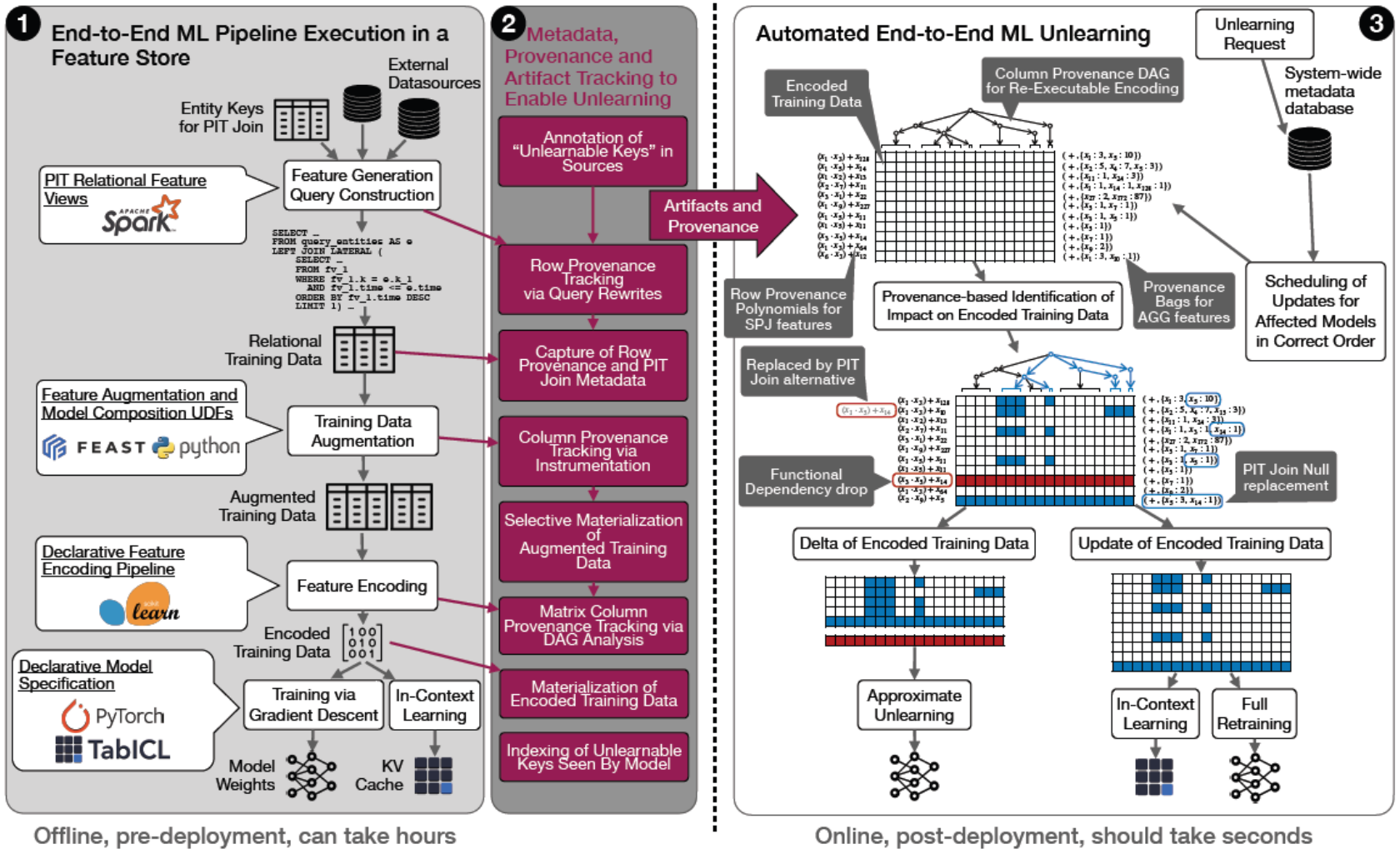
Figure 6: Data layout selection workflow.

$$\text{Minimize } C_W = \sum_{q \in W} \sum_{s \in S_q} \mathcal{D}_s - \sum_{p \in P_s} x_{sp} \cdot \mathcal{U}_{sp}^q$$



- **Data Layout Selection**
  - Avoid scanning data that are not relevant for execution
  - Find optimal partitioning strategy (e.g., fine granularity of year/month/day/hour)
  - **Binary Integration Programming** optimization for finding configuration

Bonus: Machine Unlearning through Feature Store



Modern Challenges and Open Problems

- **LLM Training**
  - Massive, weekly documented data; how to track data lineage?
  - Track licensing + ownership
- **LLM-Assisted Applications**
  - New kinds of metadata: prompts, contexts, intermediate reasoning steps
  - Track RAG sources: data freshness, consistency, conflicting sources
- **Multi-Agent / Agentic Systems**
  - Explainability and debuggability
  - LLM black box
  - Memory and state management tracking
  - Track coordination between (sub)agents

Exercises 1: Model Management Systems

Exercise 1: Lightweight Model Management System

- **Dataset Store and Versioning (3.0 pts)**
  - Create a dataset store with dataset versions
  - Track version ID, source, preprocessing steps; Reproducible
- **Model Registry (3.0 pts)**
  - Train and register models
  - Track metadata necessary to reproduce
- **Model Creation Lineage (2.0 pts)**
- **Model Selection under Budget (2.0 pts)**
- **Implementation**
  - API for logging, registering, and querying
  - Your choice of programming language
- **Papers for Inspiration**

https://people.eecs.berkeley.edu/~matei/papers/2016/hilda_modeldb.pdf
https://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper18.pdf

```
1_Model_registry_and_selection/
├── requirements.txt
├── solution.ipynb          # main notebook orchestrating the pipeline
├── data/
│   ├── raw/                # original dataset
│   └── ds_v*/              # versioned datasets (train.csv, test.csv, metadata.json)
├── models/
│   └── model_*/            # serialized model artifacts (e.g. .joblib)
├── registry/
│   └── model_*.json        # one metadata file per registered model
└── scripts/                # reproducible Python scripts
    ├── create_datasets.py
    ├── train_models.py
    ├── select_model.py
    └── summarize.py
```

Summary

- Parallels between model, data, and feature stores: All manage versioned ML artifacts
- Model management systems for model versioning and metadata tracking
- External and system-internal metadata tracking
- Coarse- and fine-grained lineage tracing
- Feature Stores and Point-in-time join
- Open challenges

Please read the recommended papers