Enhancing Transparency and Reproducibility in AI Model Training through Provenance-Enabled Data Preprocessing and Workflow Documentation

Nils Hoffmann DLR – SP
Institut für Softwaremethoden zur Produkt-Virtualisierung
nils.hoffmann@dlr.de

DLR

Motivation

- AI models increasingly influence critical decision making
- Transparency often lacking
  - In model, data and workflow

- Workflow transparency
  - Understand assumptions and limits
  - Reproducible research

- Training workflows increasingly more complex
  - Feature Engineering, Augmentations
  - Large amounts of scraped unlabelled data
  - Automatic curation workflows
  - Versioning of data / updated measurements in real world use cases

Provenance in Data and Workflows

- Provenance: Tracking origin and transformation of models and data in ML Workflow

- Transparency
  - Record processing and algorithms used
  - Document assumptions and limitations

- Reproducibility:
  - Precise replication as entire workflow is documented

- Insight into process
  - Debugging
  - Monitoring
  - Optimization

Existing Solutions

- Pachyderm
  - Provenance in Data & pipelines
  - Aimed at static production use cases, less scientific/exploratory work
- TFX
  - Schema Validation, Data Anomaly Detection
  - Little provenance recording, aimed at production use cases
- MLFlow
  - Focus on experiment management, less workflow provenance
- DataVersionControl (DVC)
  - Can record data/script dependencies
  - No in process workflow tracking

Goals and Focus

- Existing tooling is focused on established production workflows, orchestration, etc.
- Provenance of experiments/data, not workflows

- This work: Focus on (data) scientist use cases
  - Explorative model and workflow development
  - Jupyter Notebooks, disaggregated processing steps

- Tooling for low overhead provenance recording
- Additional developer value
  - Metadata recording
  - Monitoring
  - Debugging, Sampling

Overall

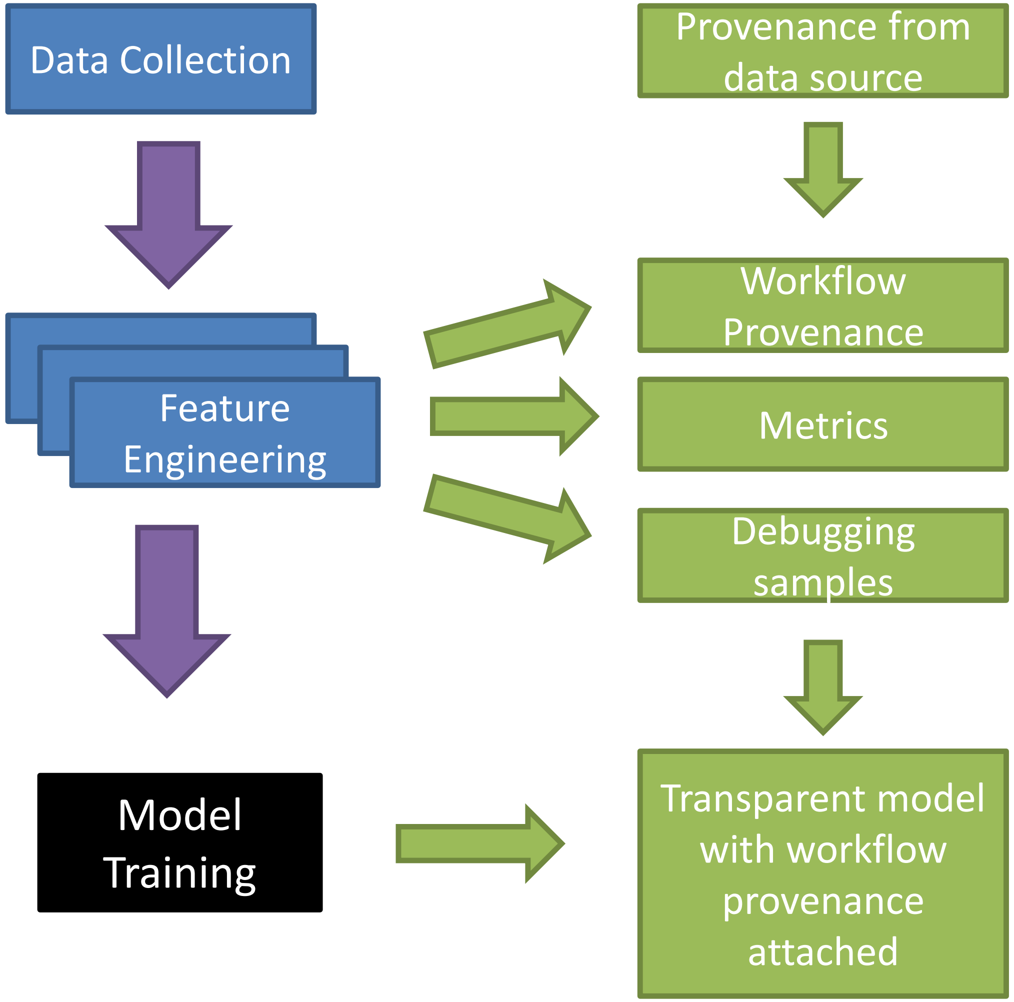
- Feature Engineering and Model Training Stages

- Feature Engineering:
  - Batch Processing (e.g. normalization factors)
  - Stream Processing (e.g. data augmentation)

- Same operations on all samples

  - Record structure of workflow
  - Record metrics stochastically

  - Low overhead



Implementation

- Pure Python library
- API similar to Tensorflow data

- Dataset as core element implements chainable operations
  - Creation form utility methods or arbitrary generators
  - Implements elementwise transform (map), reductions, shuffle, batching

- Optional user supplied labels for each operation

- Dataset records provenance during execution
  - DAG structure -> Processes and intermediate Artifacts
  - Schemas
  - Samples
  - Execution performance
- Record environment (e.g. package versions)

Simple computer vision use case demo

- Load training data from various folders
- Label with class
- Shuffle together
- Clean data
- Random augment
- Batch for training

```
ds = datasets.load_from_dir(folder, identifier=f"Load class {label}")
ds = ds.map(lambda img: (img, label), "add label identifier to images")

ds = ds.map(lambda sample: TrainingSample((sample.image - mean) / std, sample.label),
                 identifier="Normalize data")

shuffle_generator = np.random.default_rng(seed=123)
shuffled = ds.shuffle(shuffle_generator, buffer_size=4096, identifier="Shuffle samples") \
    .batch(hparams.batch_size, identifier="Batch data")
```

Monitoring

- Workflow structure: Operators and metdata
- Samples from data itself
- Provided through REST endpoint while pipeline is running

- Low performance overhead

Simple computer vision use case demo

Processes

0 **Load class 0**

Average execution time per item: N/A

1 **add label identifier to images**

Average execution time per item: 2.4µs

2 **Load class 1**

Average execution time per item: N/A

3 **add label identifier to images**

Average execution time per item: 2.6µs

4 **Load class 2**

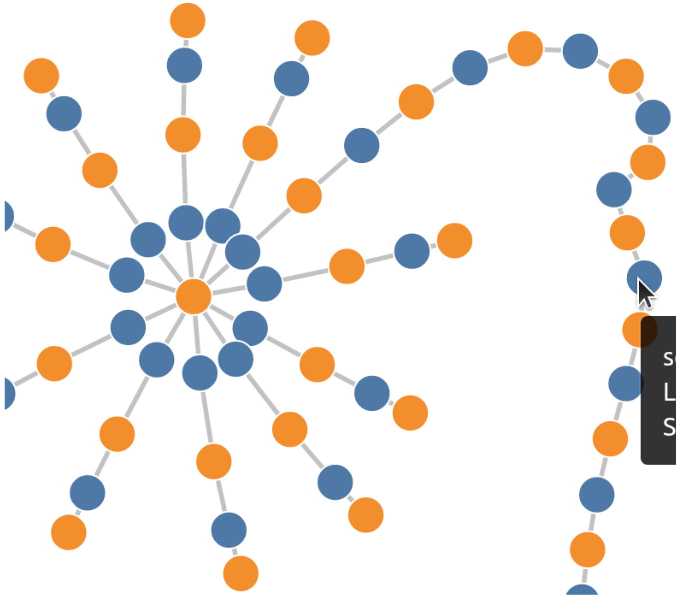
Average execution time per item: N/A

5 **add label identifier to images**

Average execution time per item: 3.0µs

6 **Load class 3**

Provenance Graph



Artifact Streams

0 **File contents of datasets/imagenette2-320/train /n03417042**

Iterator length: 961

1 **add label identifier to images**

Iterator length: 961

consumer IDs: 41	producer IDs: 3	Shapes: [('PosixPath', 'int')]

2 **File contents of datasets/imagenette2-320/train /n03028079**

Iterator length: 941

3 **add label identifier to images**

Iterator length: 941

4 **File contents of datasets/imagenette2-320/train**

Conclusion & Outlook

- Tooling for Provenance Collection in Data Science Workflows
- Expose gathered Provenance and Metadata information for monitoring and debugging

- Provide provenance to outside dependencies with DVC
  - Link DAG Inputs/Outputs against DVC file hashes

- Monitoring at central location
  - Integration into Tensorboard using Plugin

- Extension to distributed use case
  - Collecting and merging provenance data

- Demonstration on larger scale use case