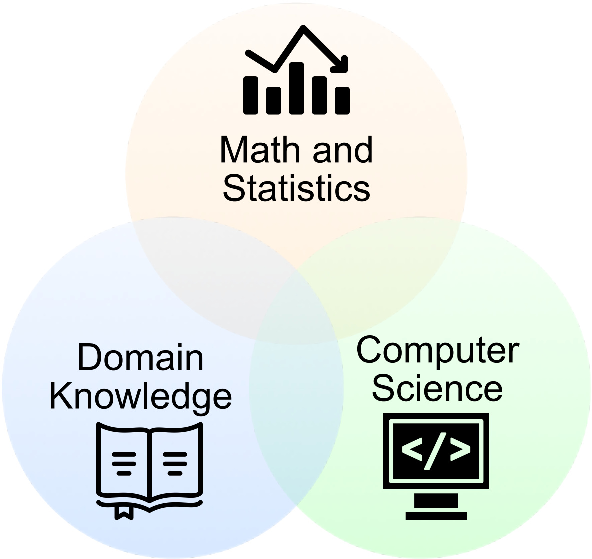
Optimizing Machine Learning Workloads in Collaborative Environments

Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Ziawasch Abedjan, Tilmann Rabl, and Volker Markl

German Research Center for Artificial Intelligence

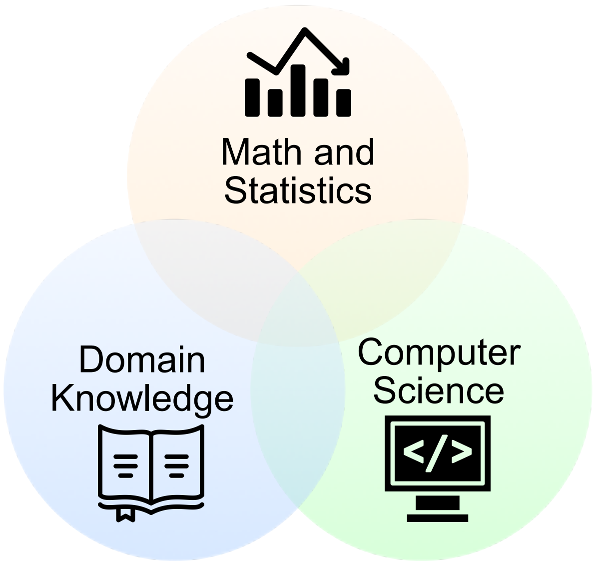
Technische Universität Berlin

Introduction



Data Science and Machine Learning

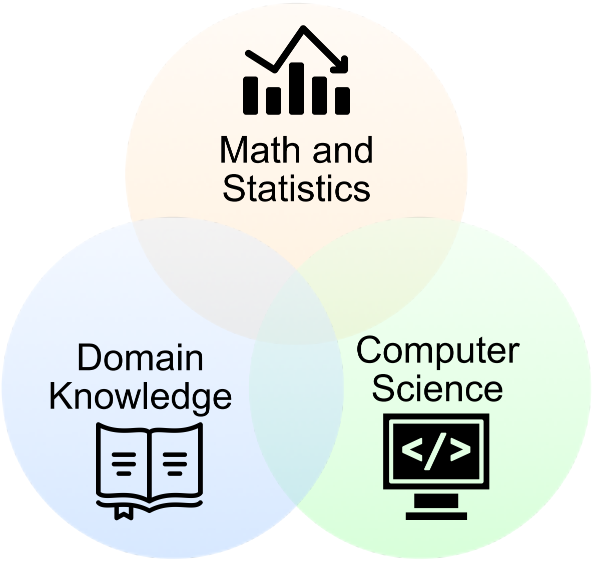
Introduction



Data Science and Machine Learning

Data Science and ML Toolkit

Introduction



Data Science and Machine Learning

Data Science and ML Toolkit

High-quality DS and ML applications require effective collaboration

Collaborative DS and ML

Collaborative DS and ML

- Jupyter Notebooks enable sharing code and results

Collaborative DS and ML

- Jupyter Notebooks enable sharing code and results

- Containerized environments enable execution and deployment of the notebooks

[1]

1. https://www.docker.com/

Collaborative DS and ML

- Jupyter Notebooks enable sharing code and results
- Containerized environments enable execution and deployment of the notebooks
- Platforms, such as Google Colab and Kaggle, enable effective collaboration among data scientists

[1]

[2]

[3]

1. https://www.docker.com/
2. https://colab.research.google.com/
3. https://www.kaggle.com/

Problems

Lack of generated data artifacts management

Problems

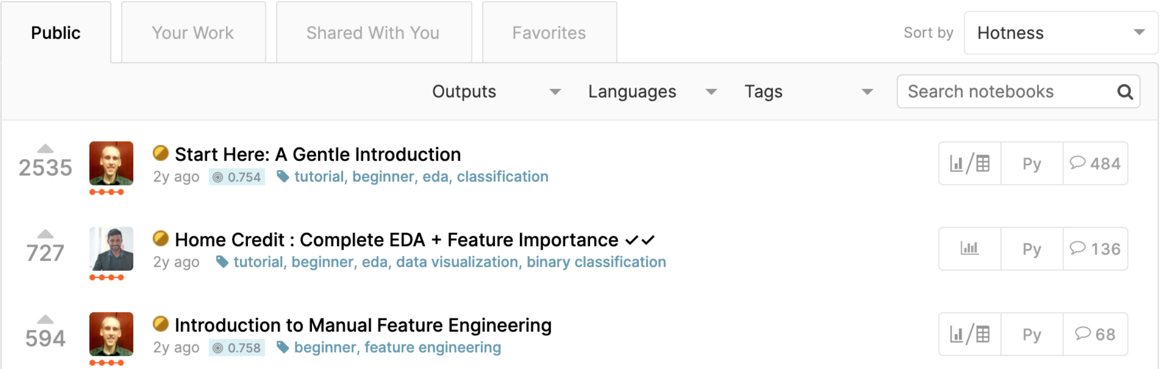
Lack of generated data artifacts management

Re-execution of existing notebooks

Problems

Lack of generated data artifacts management

Re-execution of existing notebooks



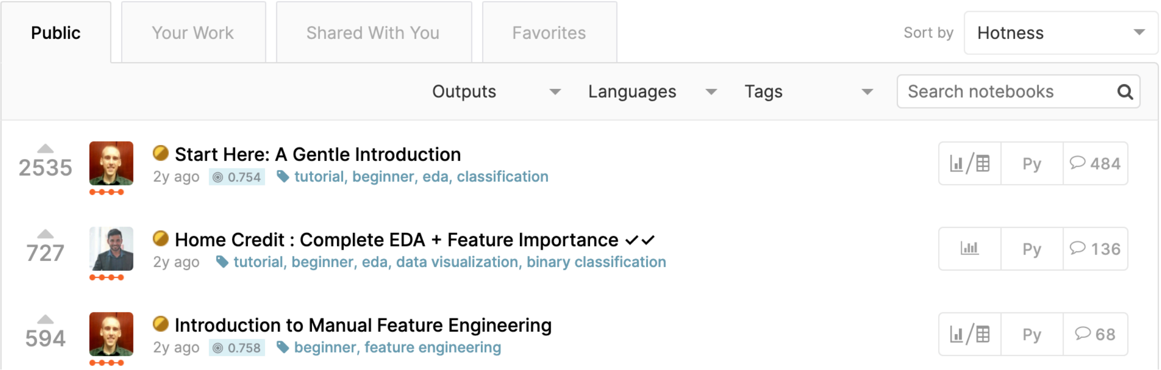
Top 3 notebooks of Home Credit Default Risk Kaggle Competition[1] generate 100s GBs of intermediate data artifact and are copied 10,000 times

1. https://www.kaggle.com/c/home-credit-default-risk

Problems

Lack of generated data artifacts management

Re-execution of existing notebooks



Top 3 notebooks of Home Credit Default Risk Kaggle Competition[1] generate 100s GBs of intermediate data artifact and are copied 10,000 times

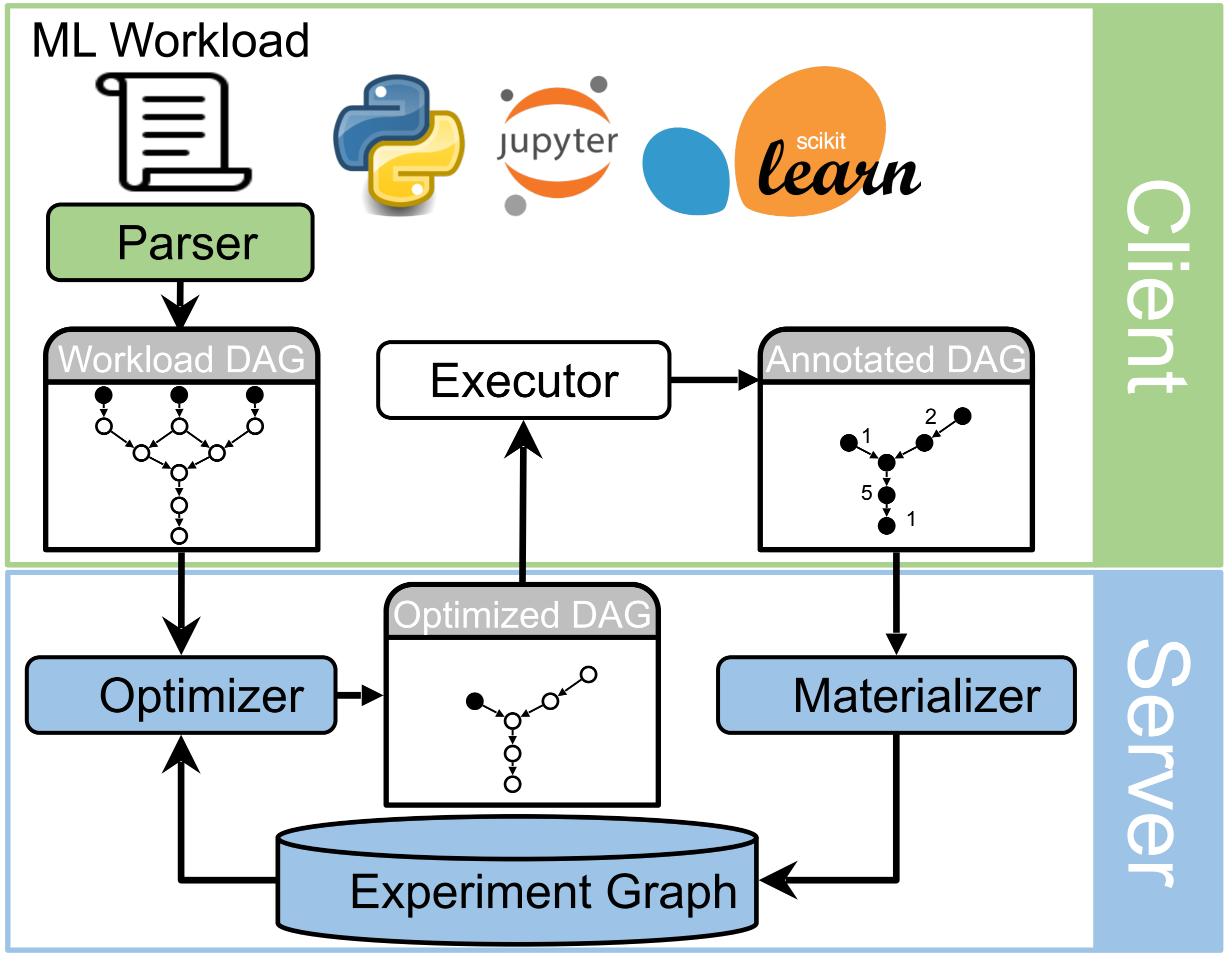
Lack of **data management** in existing collaborative environment leads to **1000s of hours of redundant** data processing and model training

1. https://www.kaggle.com/c/home-credit-default-risk

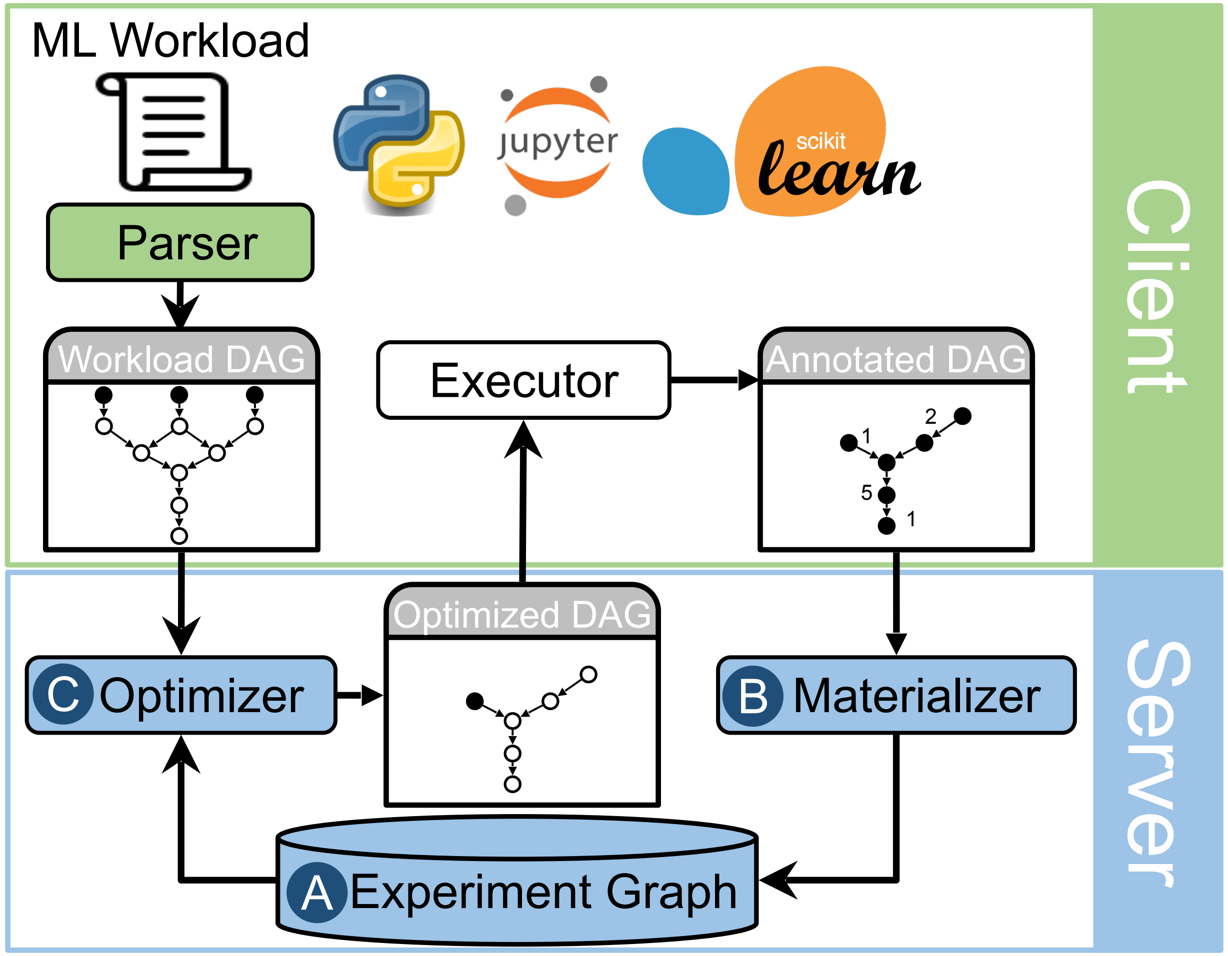
German Research Center for Artificial Intelligence

Technische Universität Berlin

Collaborative ML Workload Optimizer



Collaborative ML Workload Optimizer



Ⓐ Experiment Graph
- Union of all the Workload DAGs
- Vertices: data artifacts
- Edges: operations

Ⓑ Materializer
- Store data artifacts with high-likelihood of future reuse

Ⓒ Optimizer
- Linear-time reuse algorithm
- Finds optimal execution DAG

Materialization Problem

Given a **storage budget**, materialize a subset of the artifacts in order to minimize the **execution cost**

Materialization Problem

Given a **storage budget**, materialize a subset of the artifacts in order to minimize the **execution cost**

Challenges:

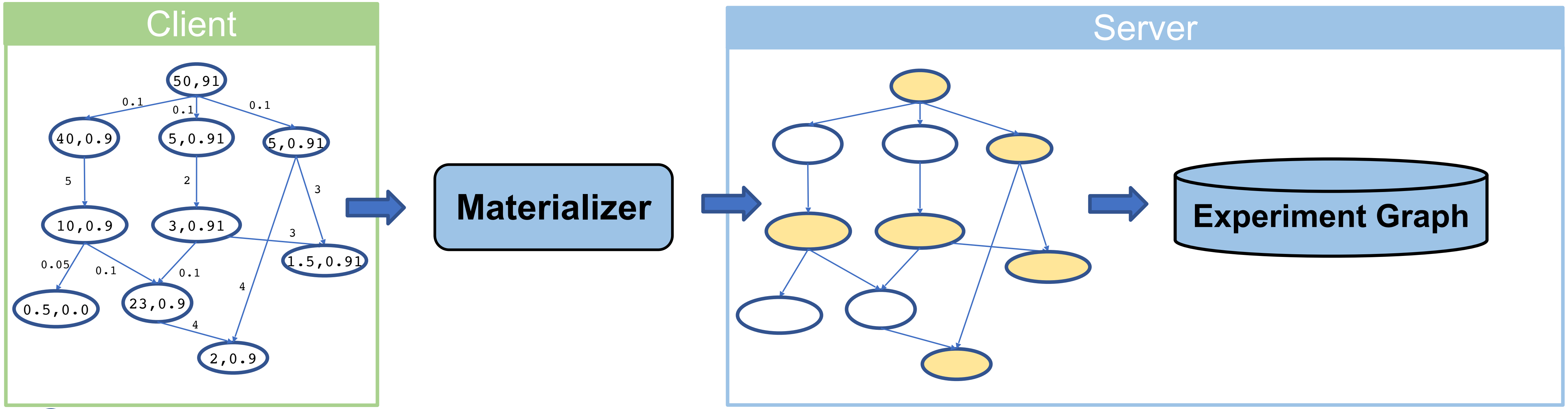
1. Future workloads are unknown
2. Even if future workloads are known a priori, the problem is NP-Hard (Bhattacherjee, 2015)
3. Accommodating large graphs and fast number of incoming workloads

Materialization Algorithm

For every artifact compute a ***utility*** value: ↓ **size** ↑ **run-time** **frequency** **potential**

Materialization Algorithm

For every artifact compute a ***utility*** value: **size** **run-time frequency potential**



Materialized artifact

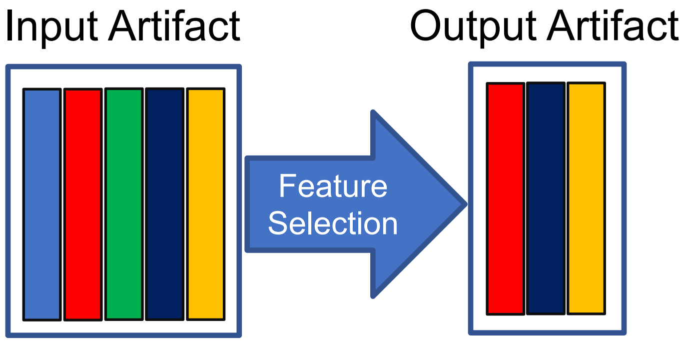
Un-materialized artifact

Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts

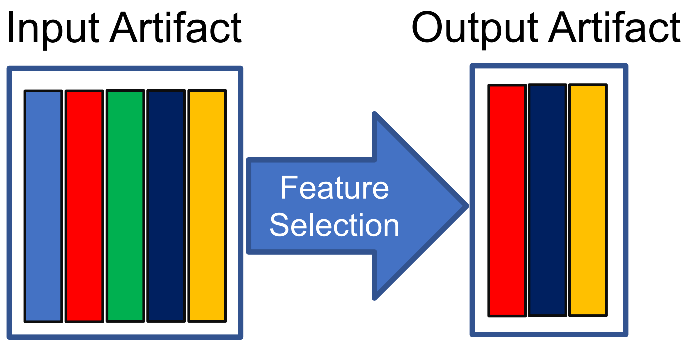
Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts
  - Feature selection operations
  - Feature generation operations



Storage-aware Materialization

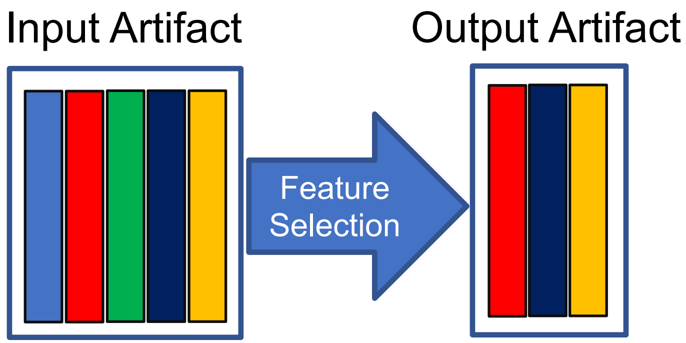
- Many **duplicated** columns in intermediate data artifacts
    - Feature selection operations
    - Feature generation operations
- Apply column deduplication strategy




```
Storage-aware Materialization
while budget is not exhausted:
    1. run materialization algorithm
    2. deduplicate the materialized artifacts
    3. update the size of unmaterialized
artifact
```

Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts
  - Feature selection operations
  - Feature generation operations
- Apply column deduplication strategy



Storage-aware Materialization
while budget is not exhausted: 1. run materialization algorithm 2. deduplicate the materialized artifacts 3. update the size of unmaterialized

Improves Storage utilization and Run-time

Reuse Problem

Given **EG** and a **new workload**, find the set of workload artifacts to **reuse** from EG and the set of artifact to **compute**

Reuse Problem

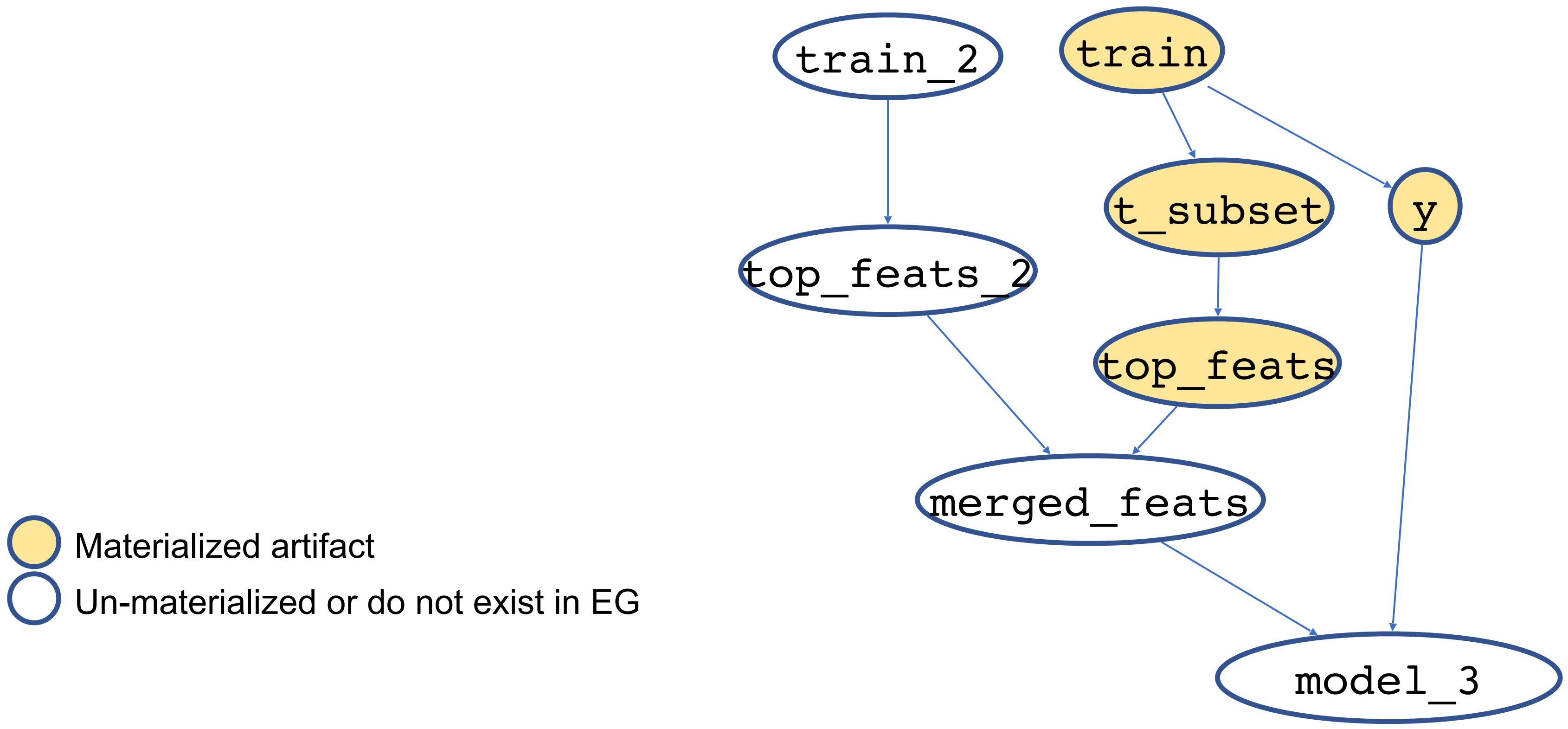
Given **EG** and a **new workload**, find the set of workload artifacts to **reuse** from EG and the set of artifact to **compute**

Challenges:

1. Exponential time complexity of exhaustive search
2. Accommodating large graphs and fast number of incoming workloads, state-of-the-art has polynomial time complexity $\mathcal{O}(|\mathcal{V}|.|E|^2)$ (Helix, 2018)

Reuse Algorithm

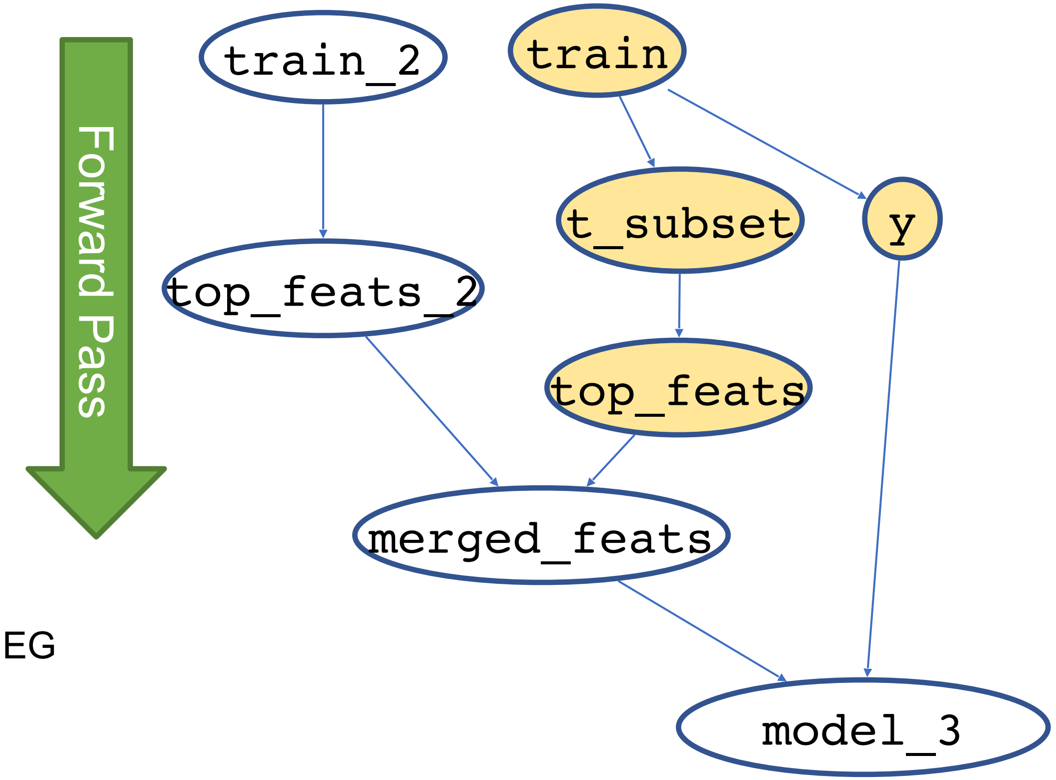
A linear-time algorithm to compute optimal execution plan with a ***forward and backward pass*** on the workload DAG



Reuse Algorithm

A linear-time algorithm to compute optimal execution plan with a ***forward and backward pass*** on the workload DAG

- Accumulate run-times
- For every vertex
  - Load or compute



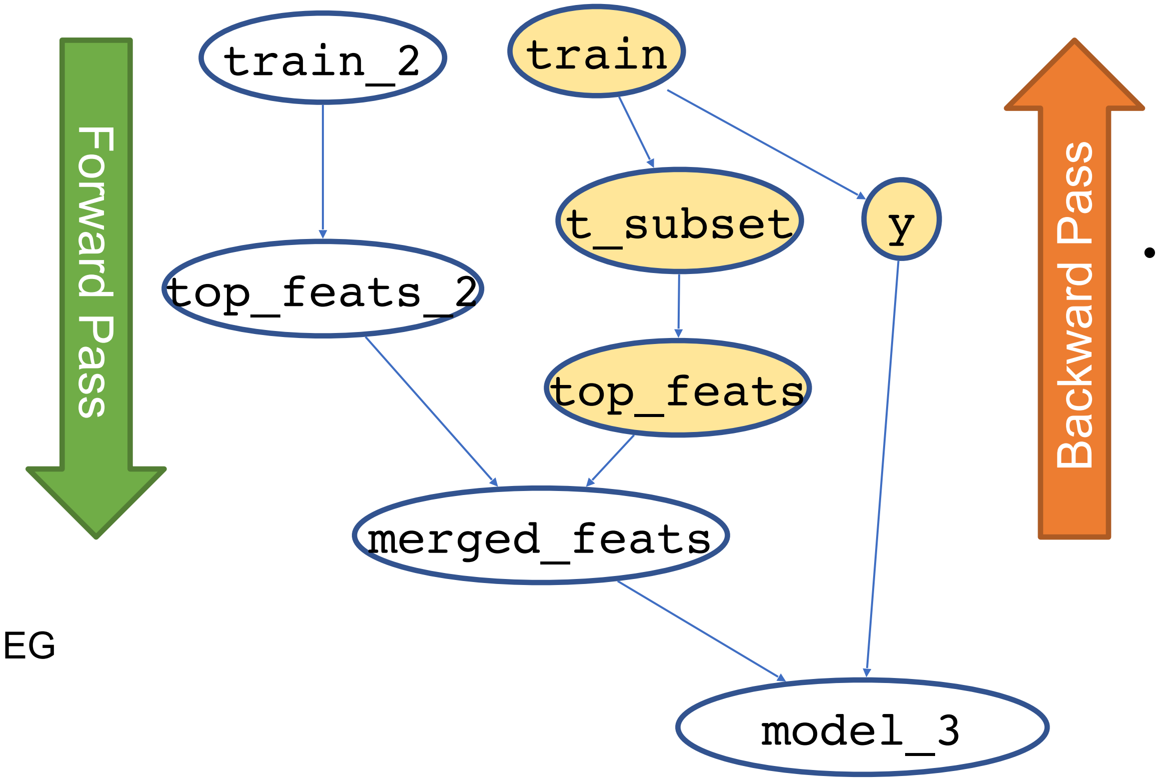
Materialized artifact

Un-materialized or do not exist in EG

Reuse Algorithm

A linear-time algorithm to compute optimal execution plan with a ***forward and backward pass*** on the workload DAG

- Accumulate run-times
- For every vertex
  - Load or compute



- Prune unnecessary loaded vertices

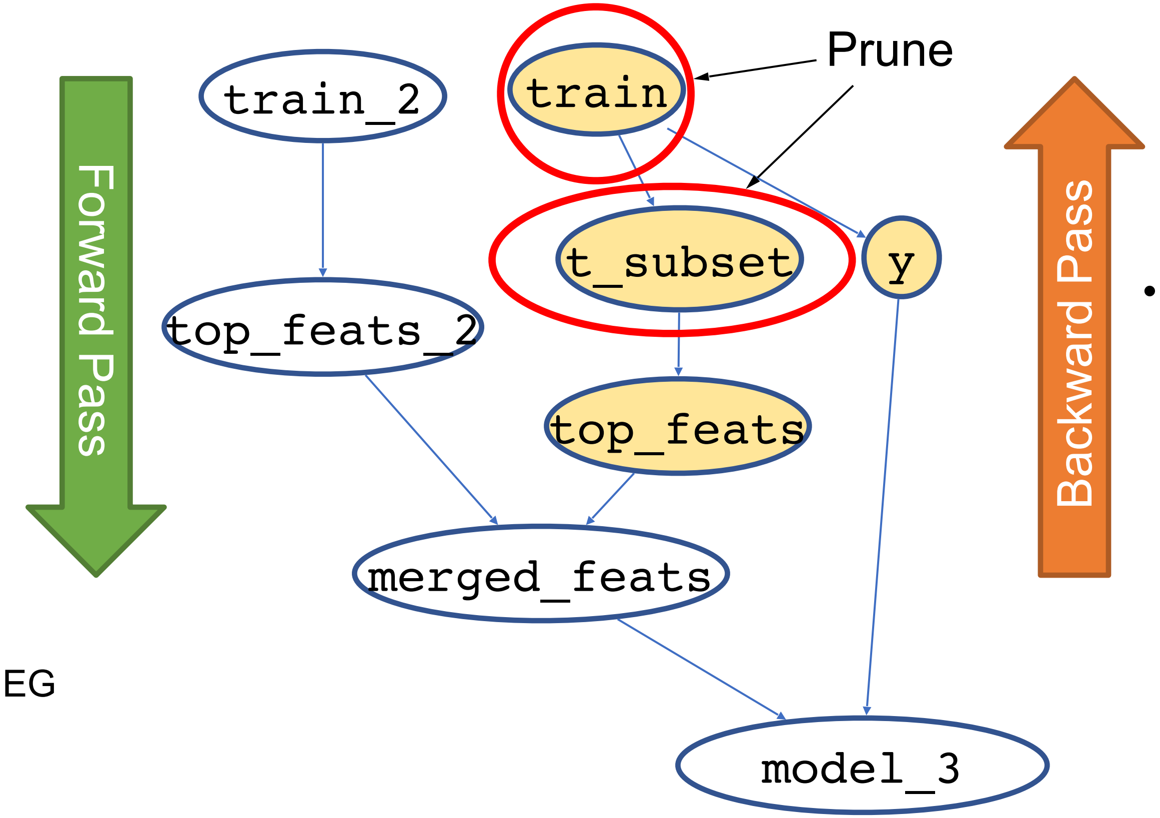
Materialized artifact
Un-materialized or do not exist in EG

German Research Center for Artificial Intelligence
Technische Universität Berlin

Reuse Algorithm

A linear-time algorithm to compute optimal execution plan with a ***forward and backward pass*** on the workload DAG

- Accumulate run-times
- For every vertex
  - Load or compute



- Prune unnecessary loaded vertices

Materialized artifact

Un-materialized or do not exist in EG

Evaluation

Workload

- **Kaggle**
  - Kaggle Home Credit Default Risk[1] Competition
  - 9 source datasets
  - 5 real and 3 generated workloads
  - Total of 130 GB artifact sizes

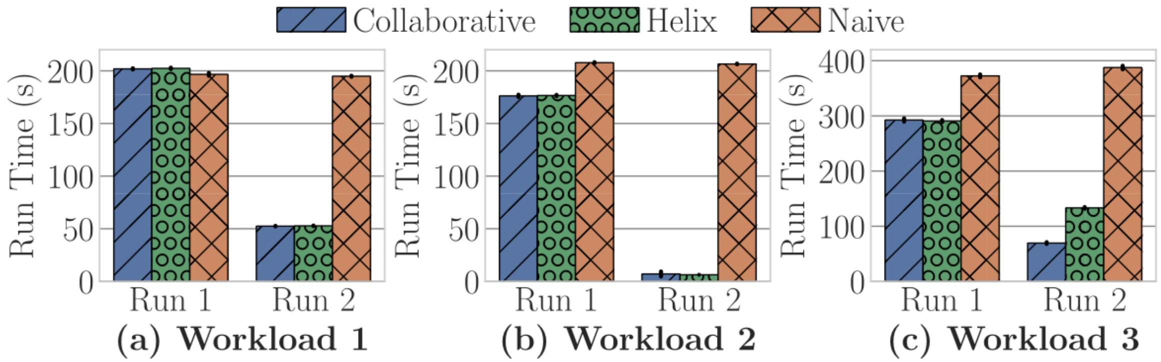
Baseline

- **Helix**
  - State-of-the-art iterative ML framework
  - Materialization Algorithm:
    - Only execution-time is considered
    - Nodes are not prioritized
  - Reuse Algorithm
    - Utilizes Edmonds-Karp Max-Flow algorithm, which runs in $\mathcal{O}(|\mathcal{V}|.|E|^2)$

- **Naïve**
  - No Optimization

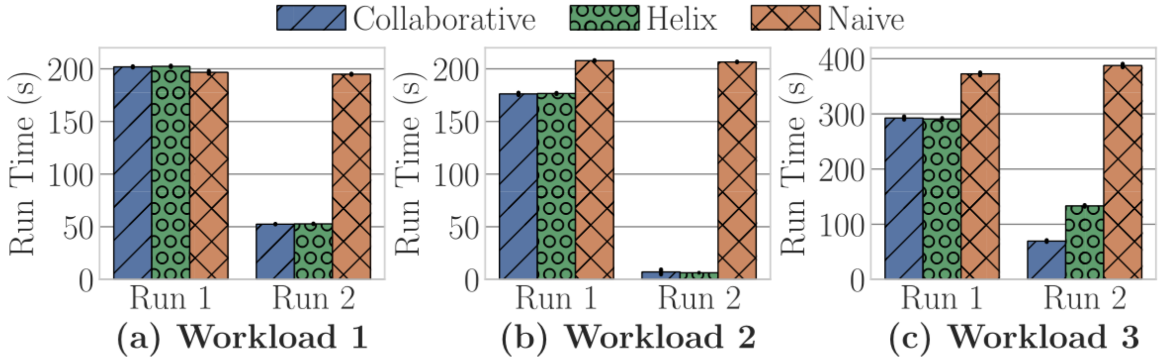
1. https://www.kaggle.com/c/home-credit-default-risk

End-to-end Run-time

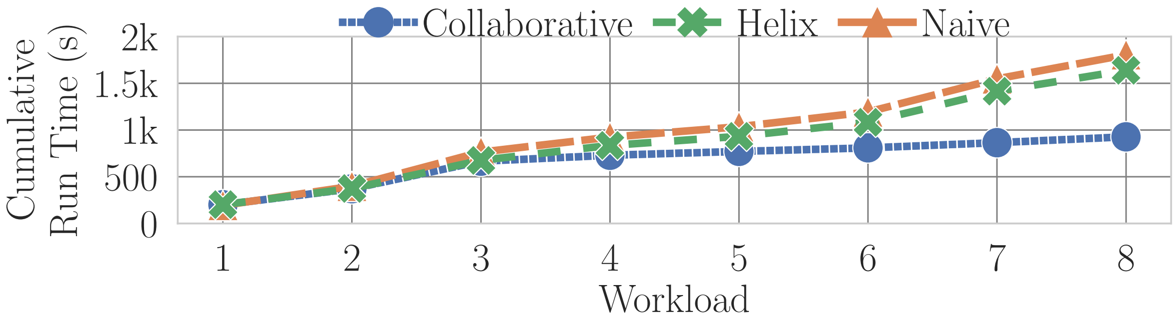


Repeated executions of Kaggle workloads (materialization budget = 16 GB)

End-to-end Run-time

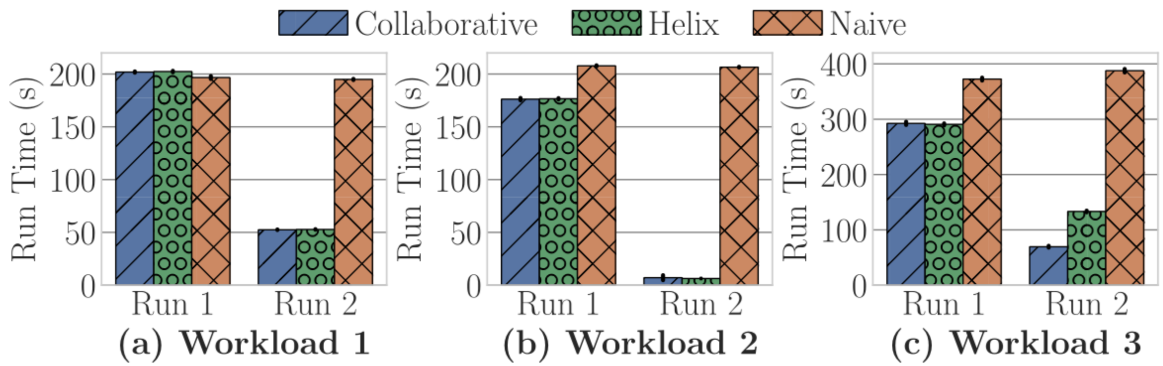


Repeated executions of Kaggle workloads (materialization budget = 16 GB)

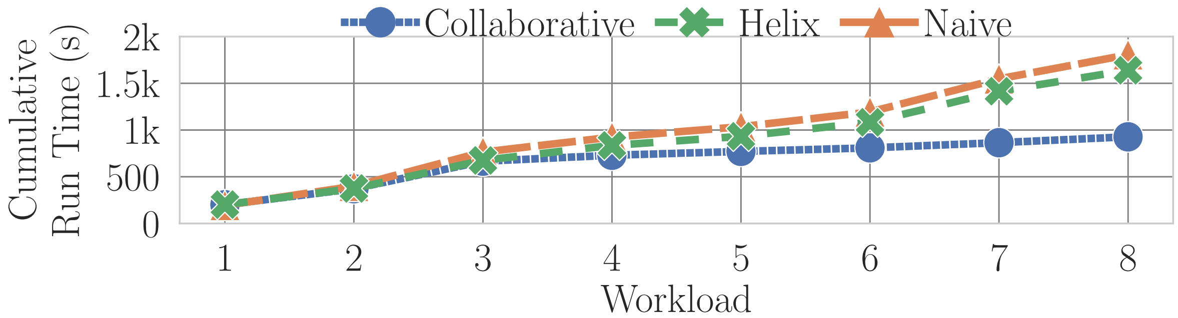


Execution of Kaggle workloads in sequence (materialization budget = 16 GB)

End-to-end Run-time



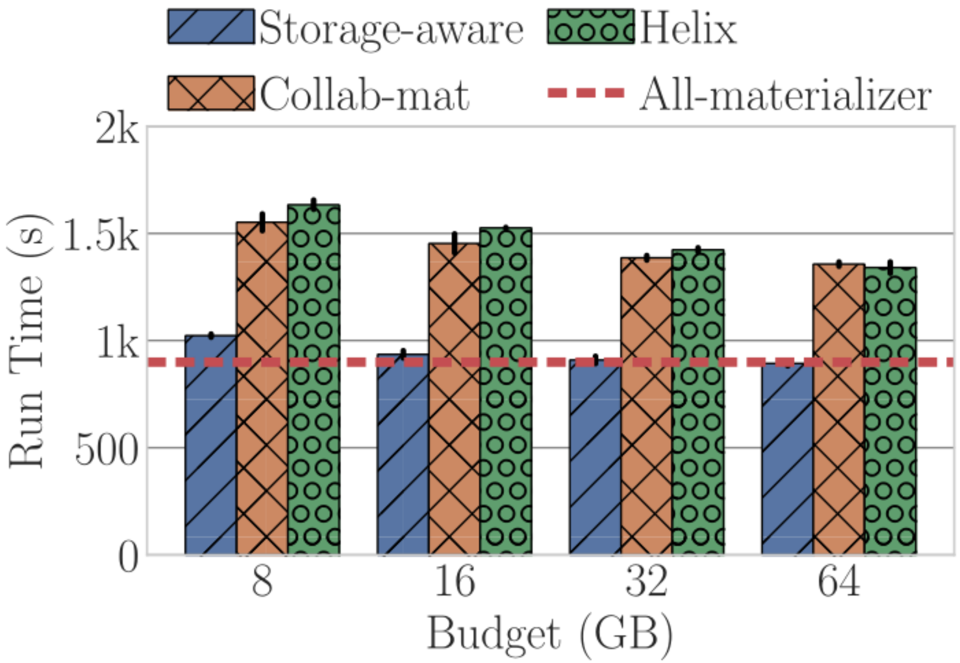
Repeated executions of Kaggle workloads (materialization budget = 16 GB)



Execution of Kaggle workloads in sequence (materialization budget = 16 GB)

Optimizing ML Workloads improves run-time up to 1 order of magnitude for repeated executions and 50% for different workloads

Materialization Impact

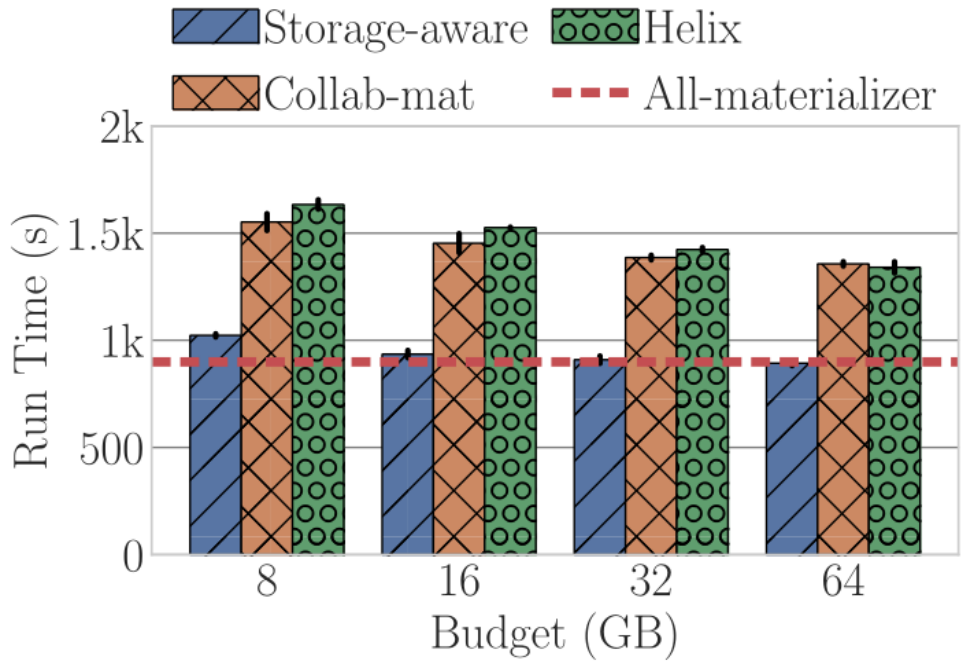


Total run-time of the Kaggle workloads with different materialization strategies and budgets

German Research Center for Artificial Intelligence

Technische Universität Berlin

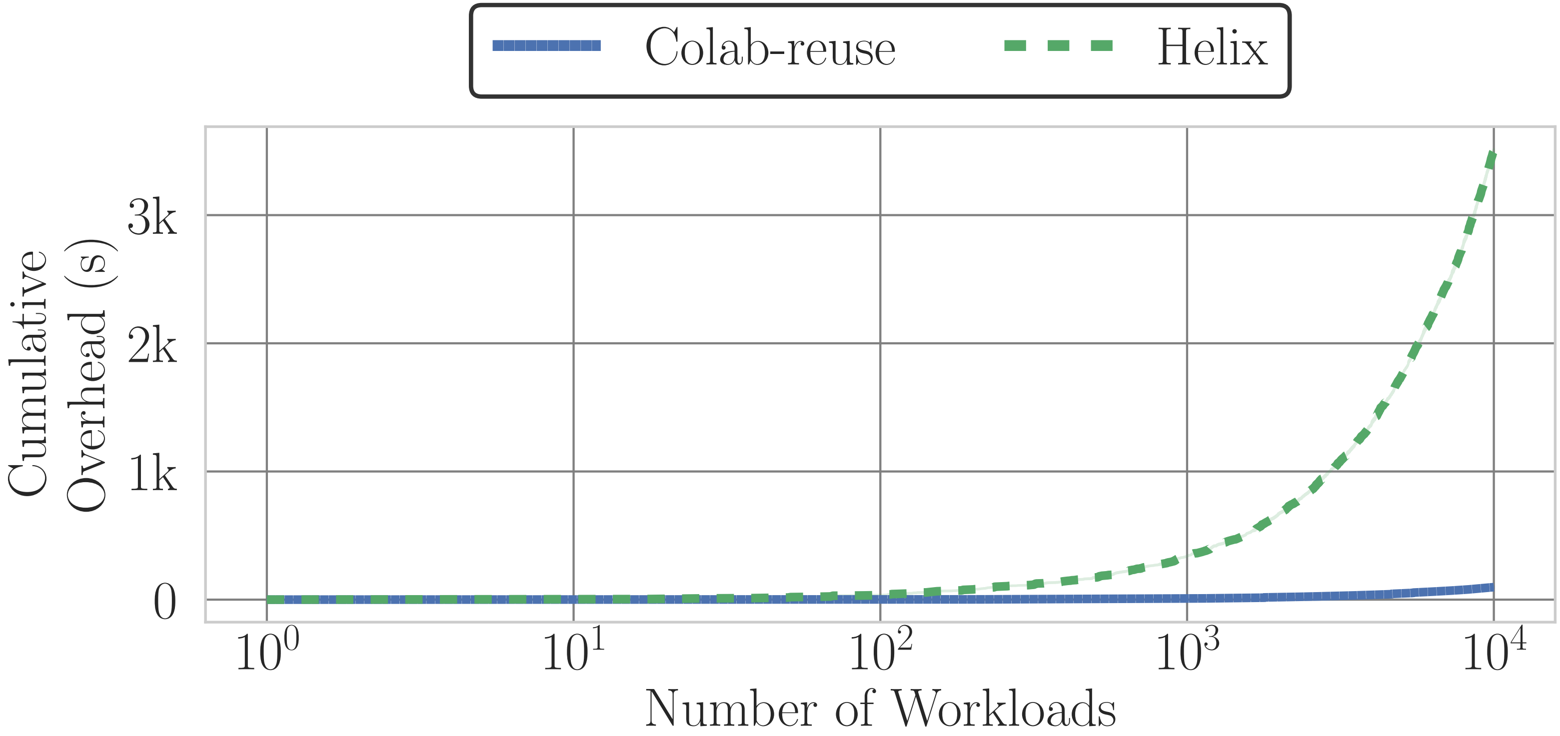
Materialization Impact



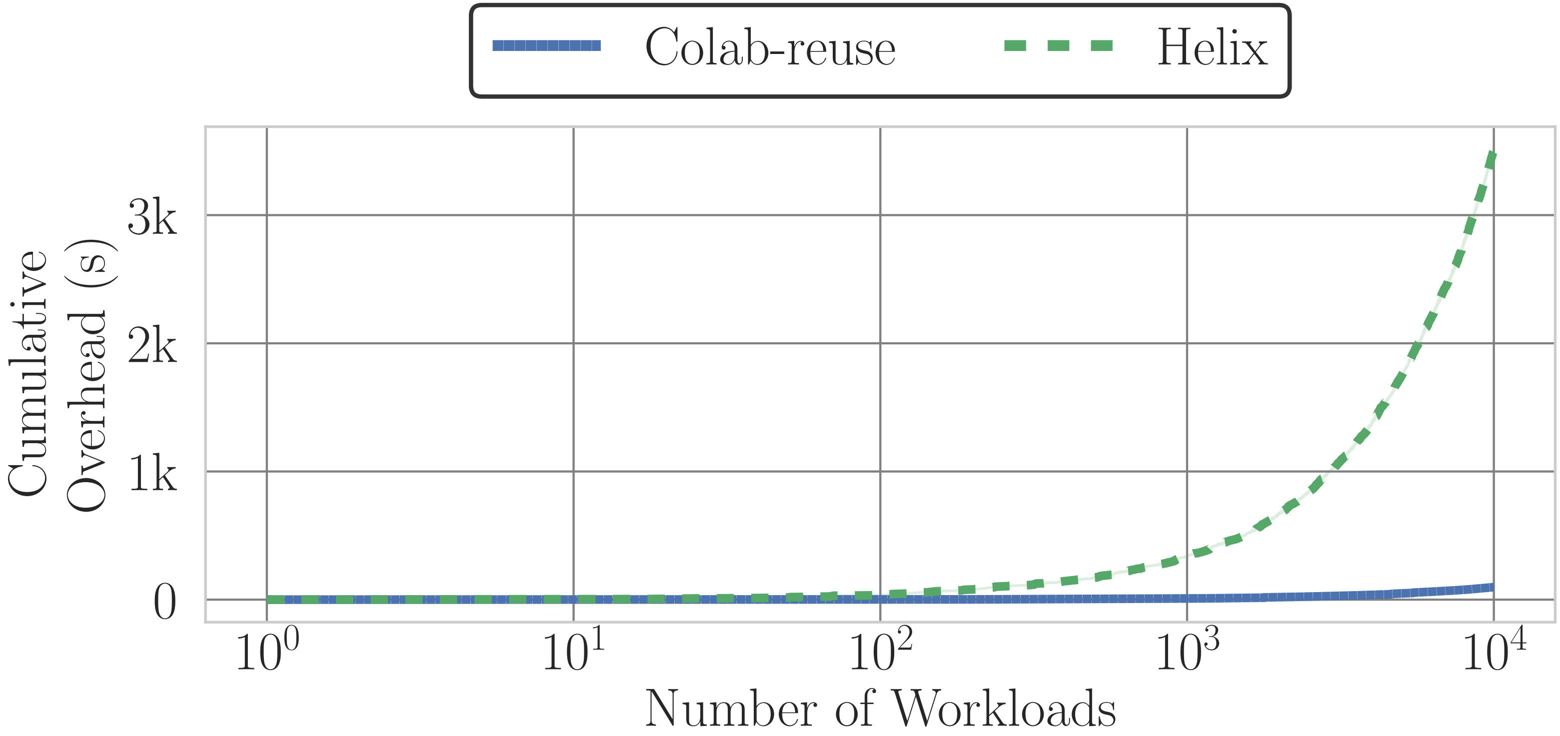
Total run-time of the Kaggle workloads with different materialization strategies and budgets

Exploiting the artifacts characteristics, such as utility and duplication rate, improves the materialization process and improves the run-time by 50%

Reuse Overhead

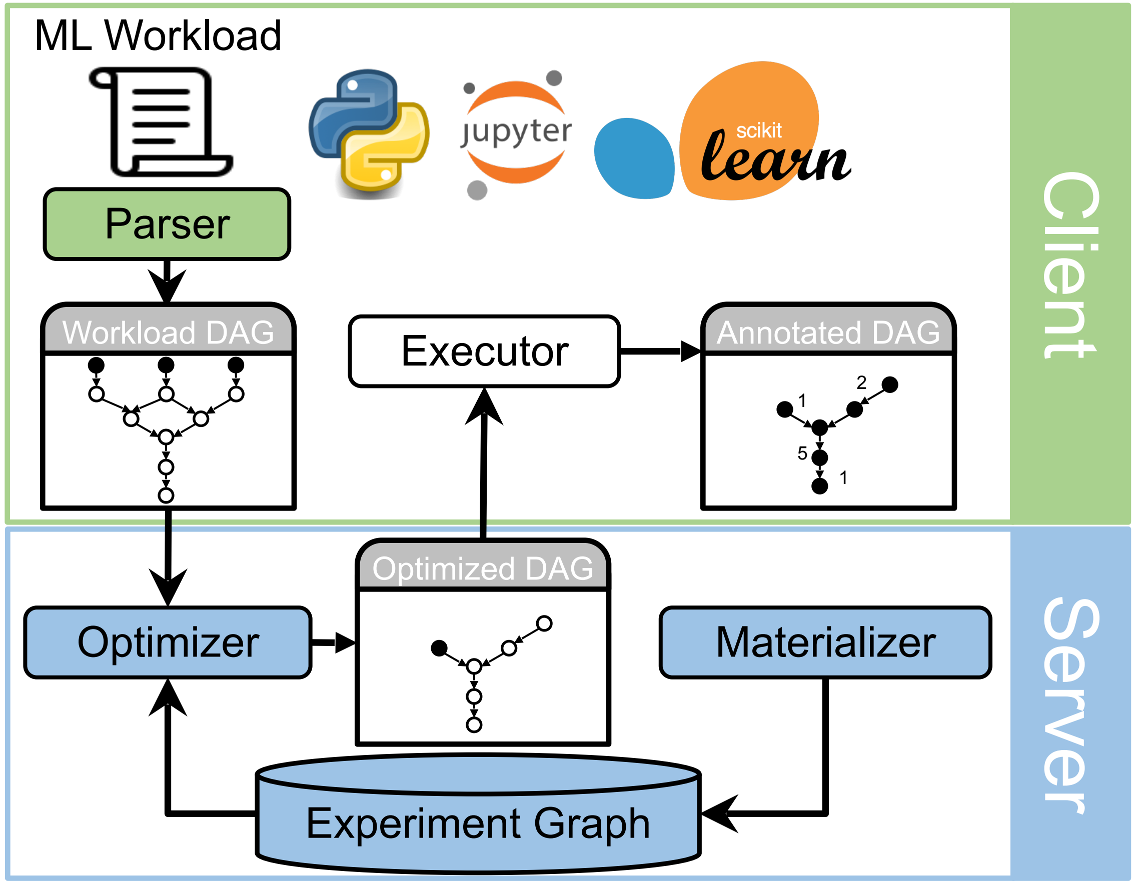


Reuse Overhead



Our linear-time reuse algorithm generates a negligible overhead in real collaborative environments where 1000s of workloads are executed

Summary



- Optimization of ML workloads in collaborative environment through ***Materialization*** and ***Reuse***, while incurring negligible overhead

- **Things not covered in the talk:**
  - API and DAG Construction
  - Quality-based Materialization
  - Model Warmstarting

References

- Bhattacherjee, Souvik, et al. "Principles of dataset versioning: Exploring the recreation/storage tradeoff." *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*. Vol. 8. No. 12. NIH Public Access, 2015.
- Xin, Doris, et al. "Helix: Holistic optimization for accelerating iterative machine learning." *Proceedings of the VLDB Endowment* 12.4 (2018): 446-460.
- Jack Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. J. ACM 19, 2 (April 1972), 248–264. https://doi.org/10.1145/321694.321699
- Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, et al. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In Posi- tioning and Power in Academic Publishing: Players, Agents and Agendas, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. *Queue*, *14*(1), 70-93.

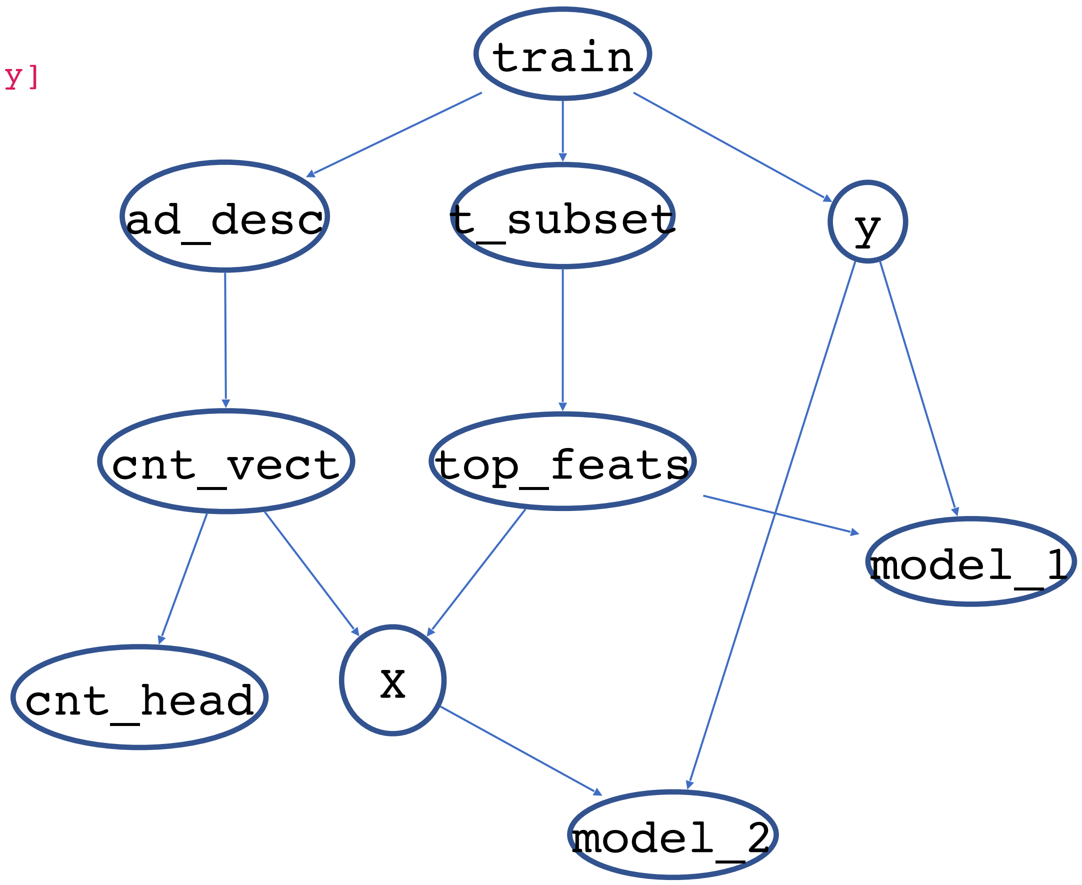
Evaluation (2)

Workload

- **Openml**
  - Classification Task 31[2]
  - 2000 scikit-learn pipelines
  - 1.5 GB of artifact sizes

DAG Construction

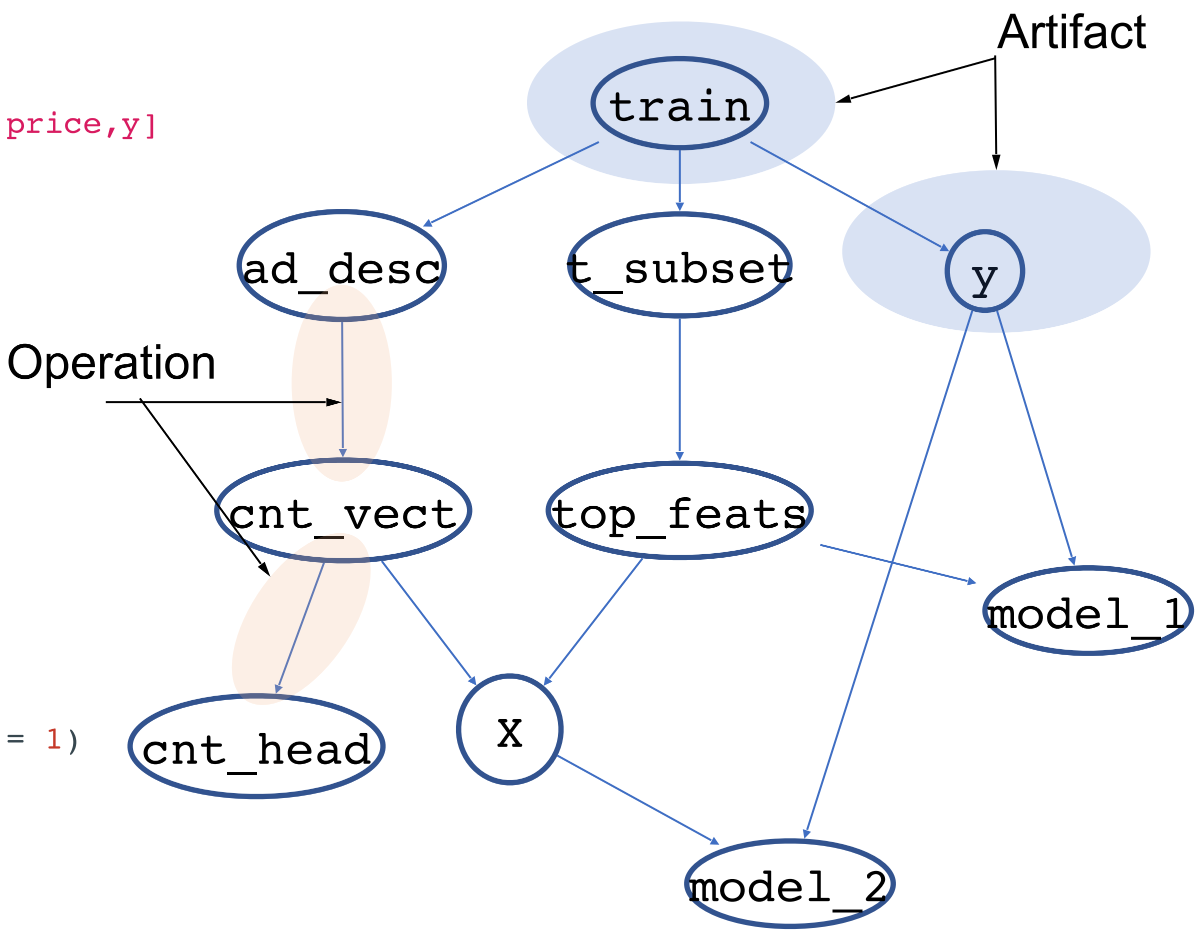
```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



German Research Center for Artificial Intelligence
Technische Universität Berlin

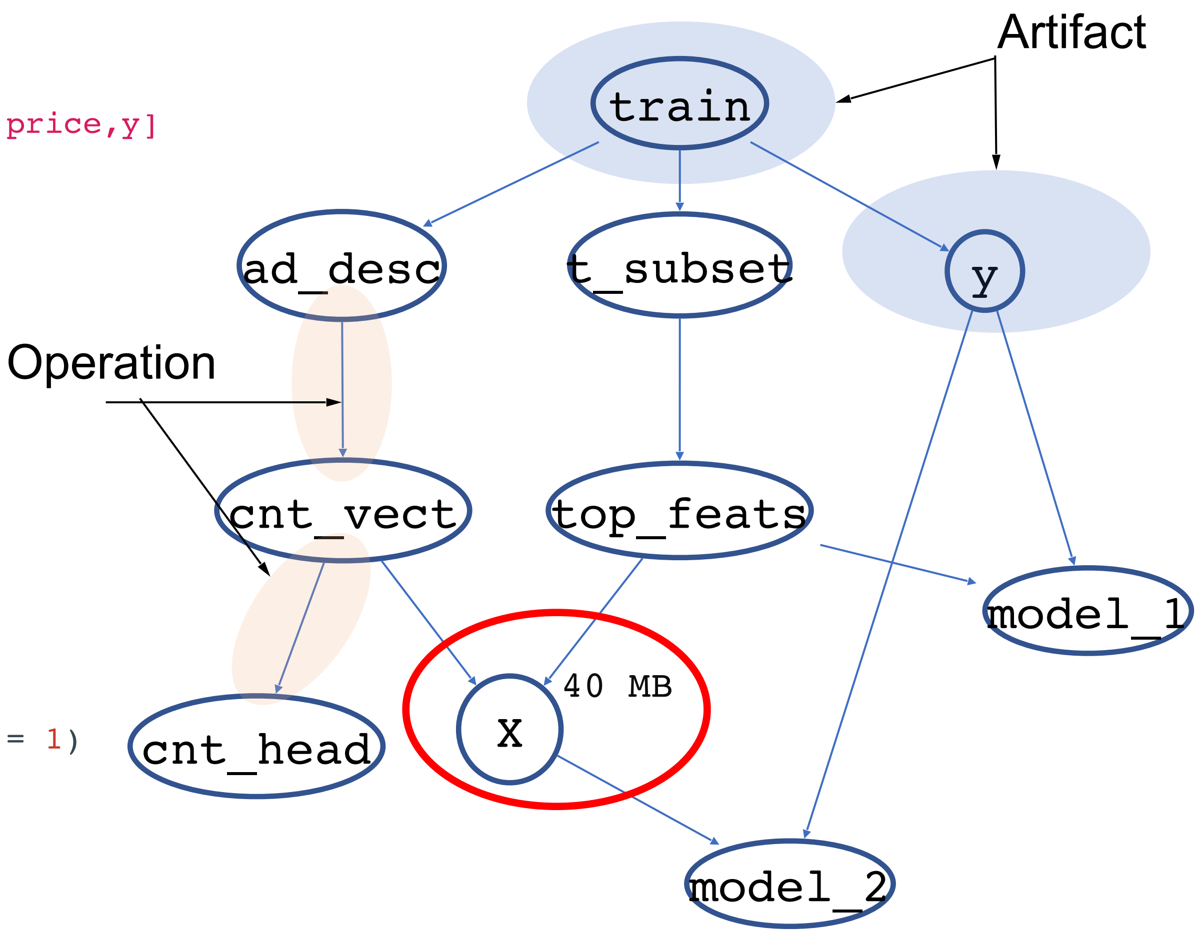
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



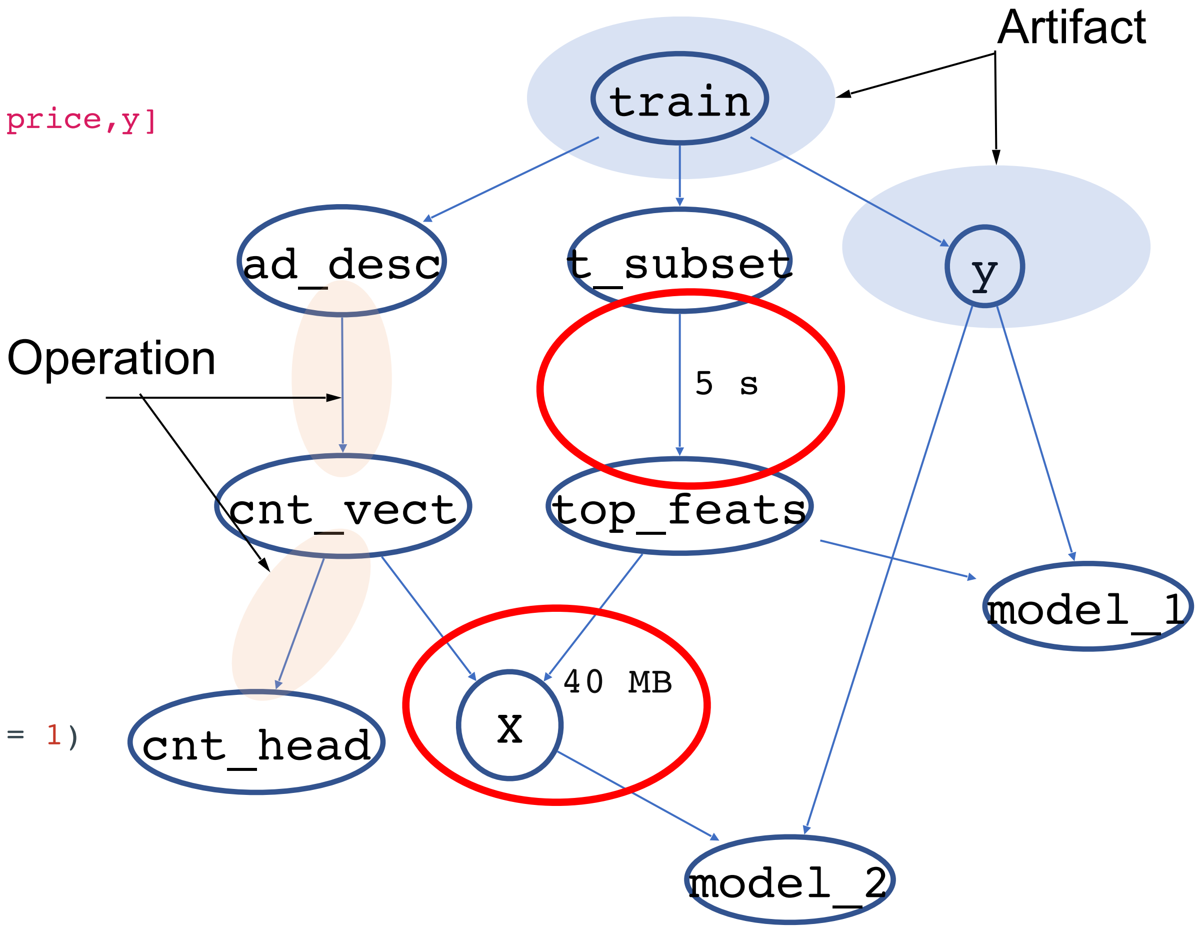
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



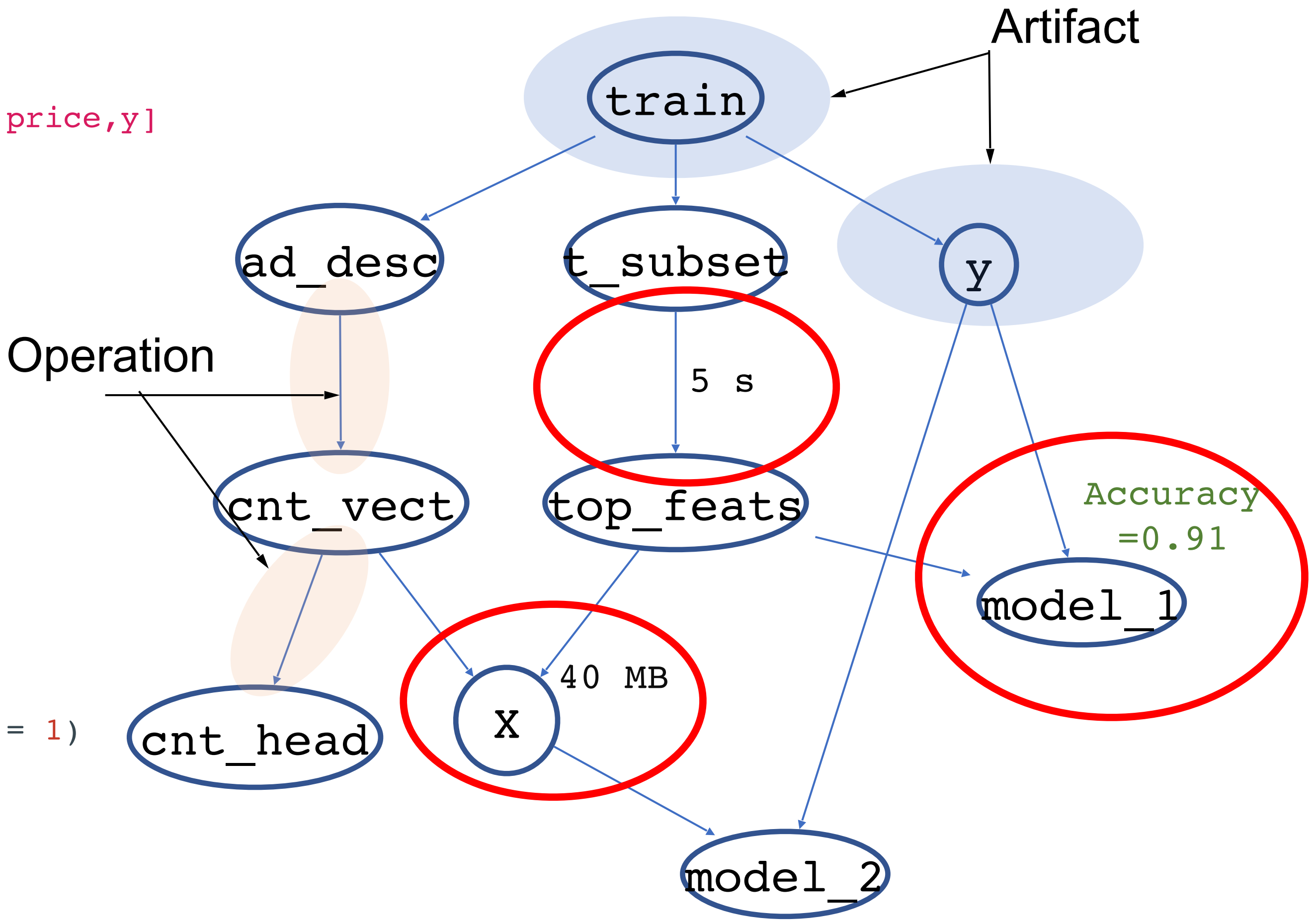
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



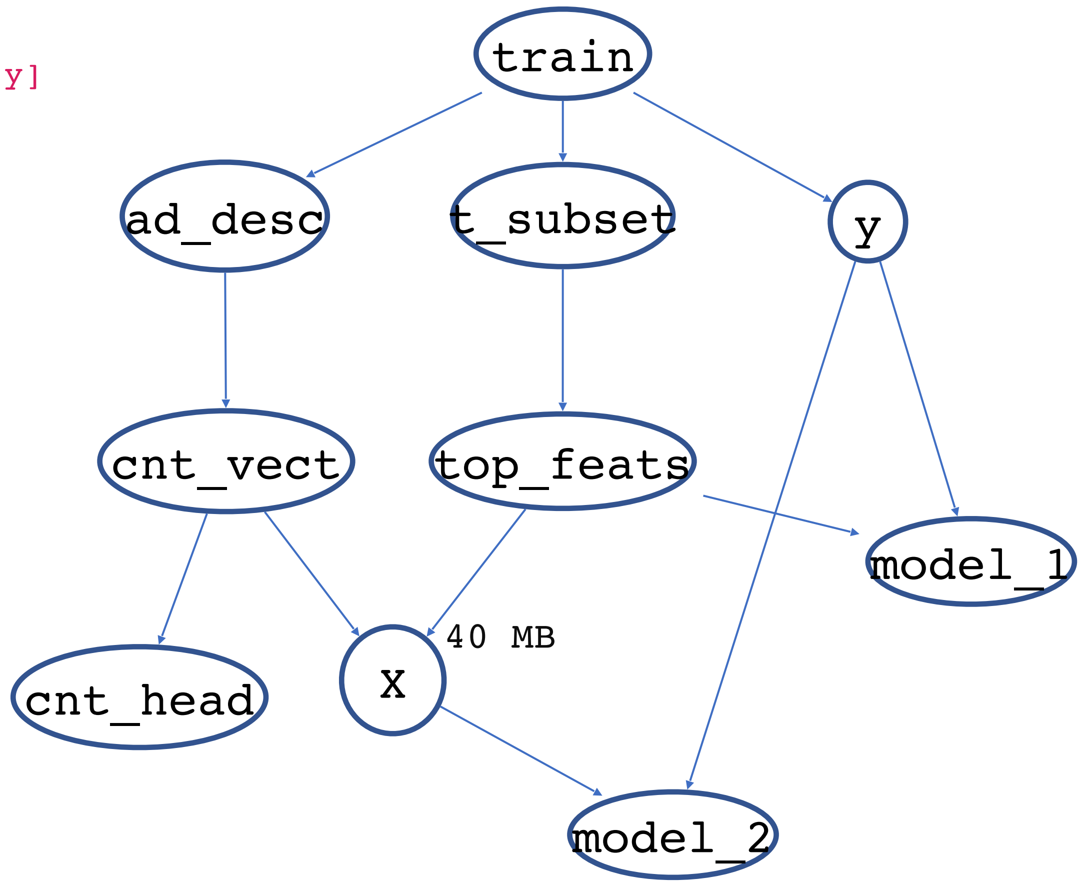
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



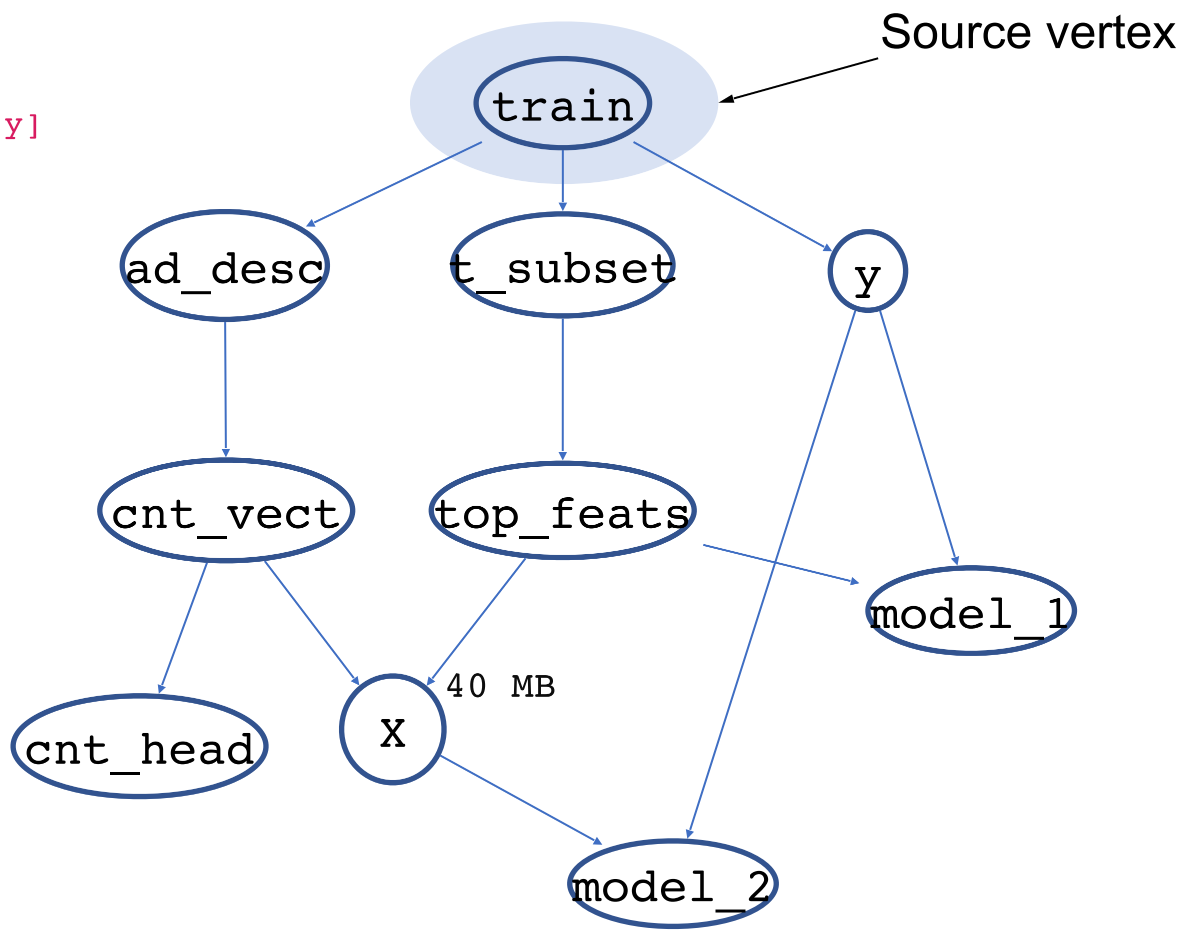
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



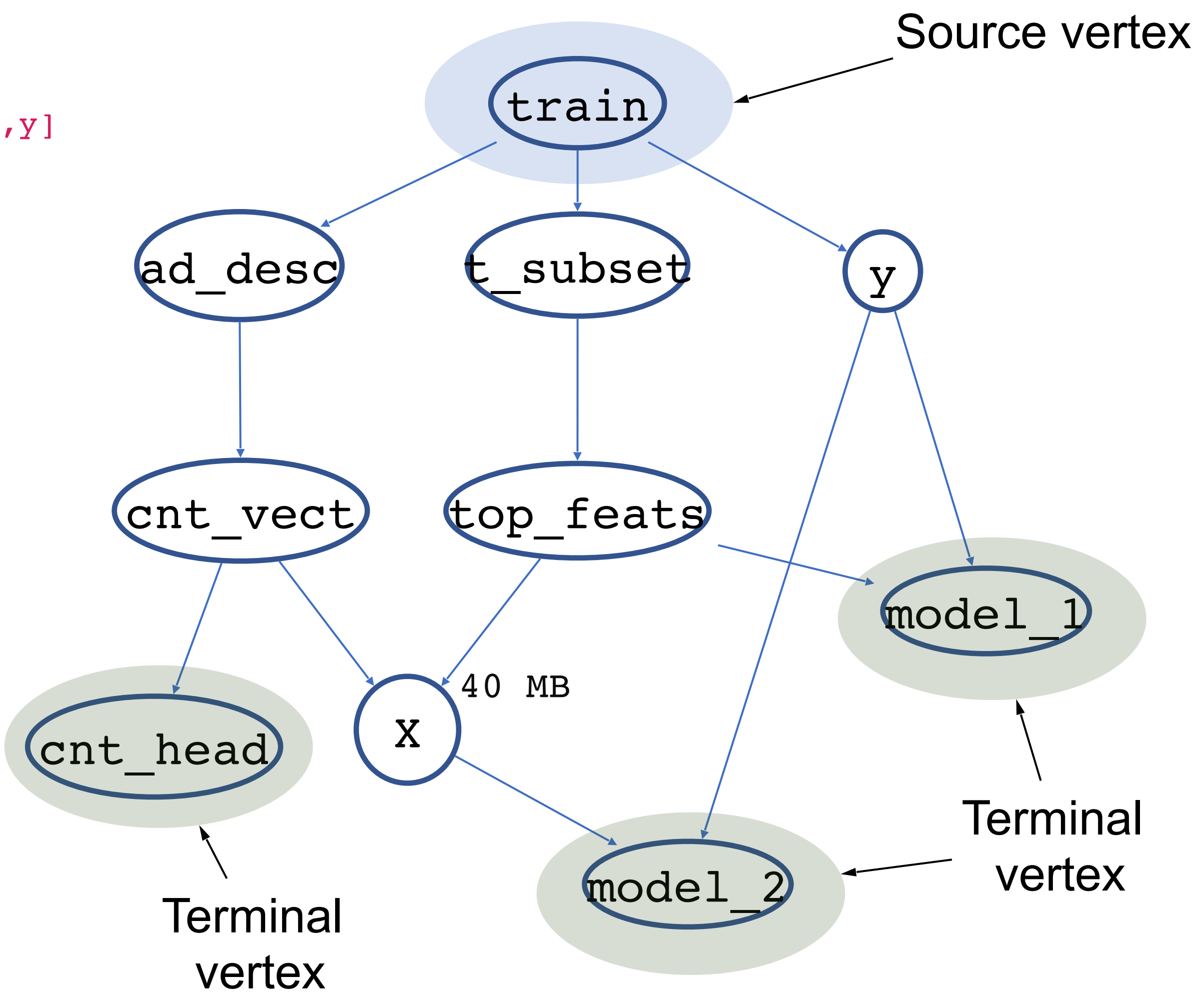
DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



DAG Construction

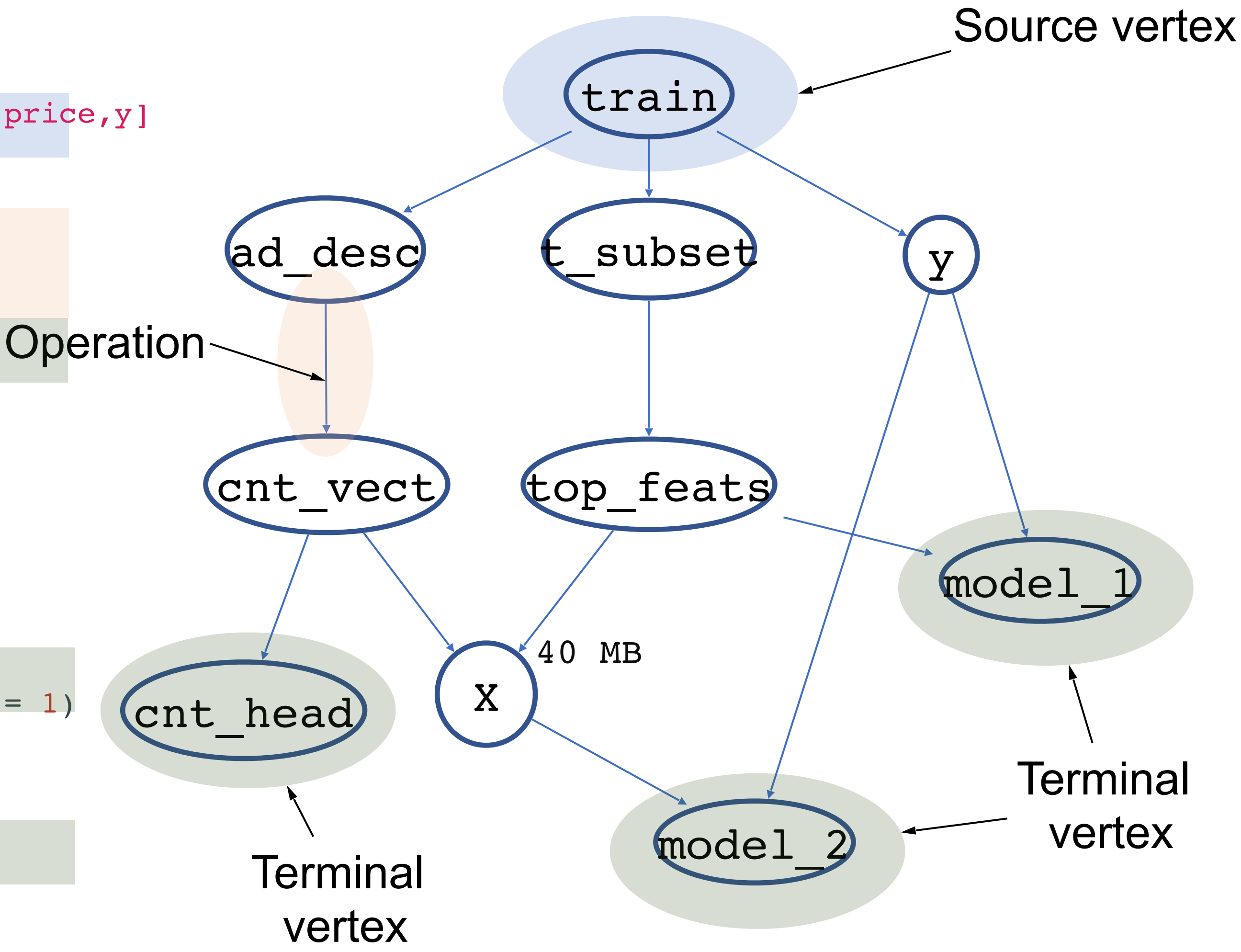
```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```



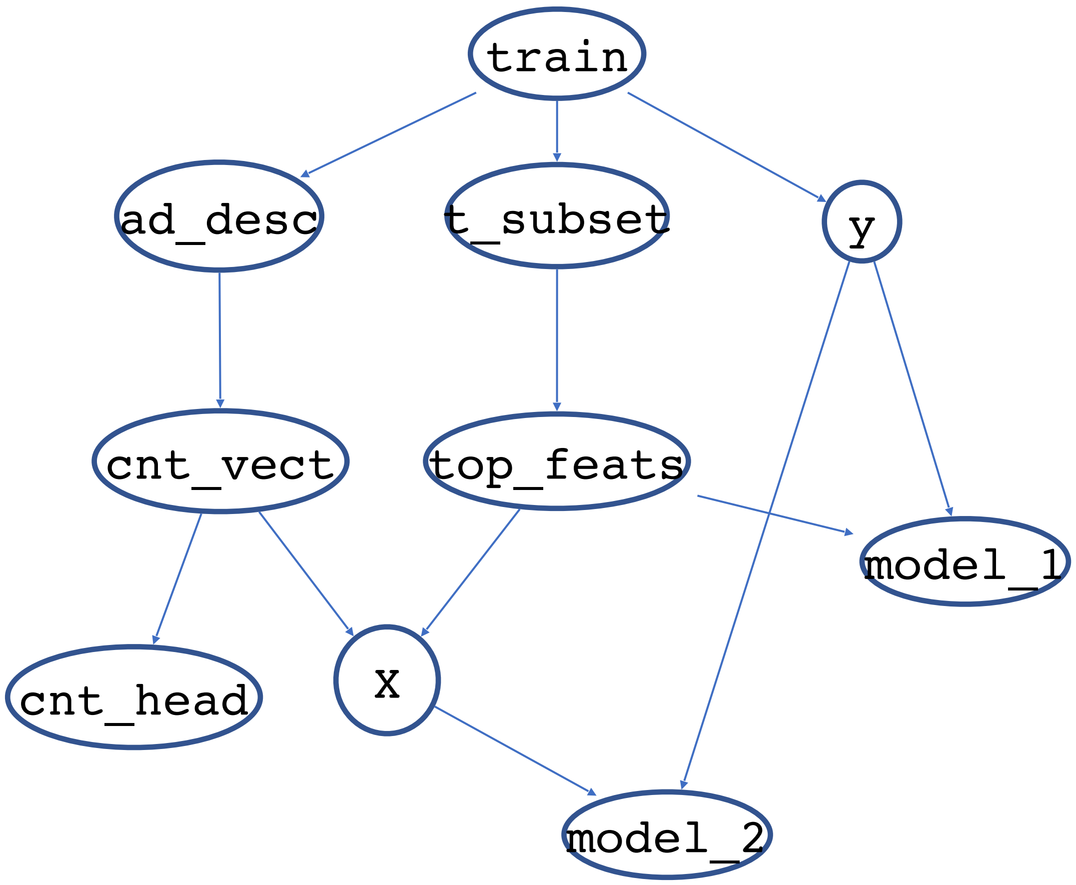
German Research Center for Artificial Intelligence
Technische Universität Berlin

DAG Construction

```
train = pd.read_csv('train.csv') # [ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc']
vectorizer = CountVectorizer()
count_vectorized = vectorizer.fit_transform(ad_desc)
print count_vectorized.head()
selector = SelectKBest(k=2)
t_subset = train[['ts','u_id','price']]
y = train['y']
top_features = selector.fit_transform(t_subset, y)
model_1 = svm.SVC().fit(top_features, y)
print model_1 # terminal vertex
X = pd.concat([count_vectorized,top_features], axis = 1)
model_2 = svm.SVC().fit(X, y)
print model_2 # terminal vertex
```

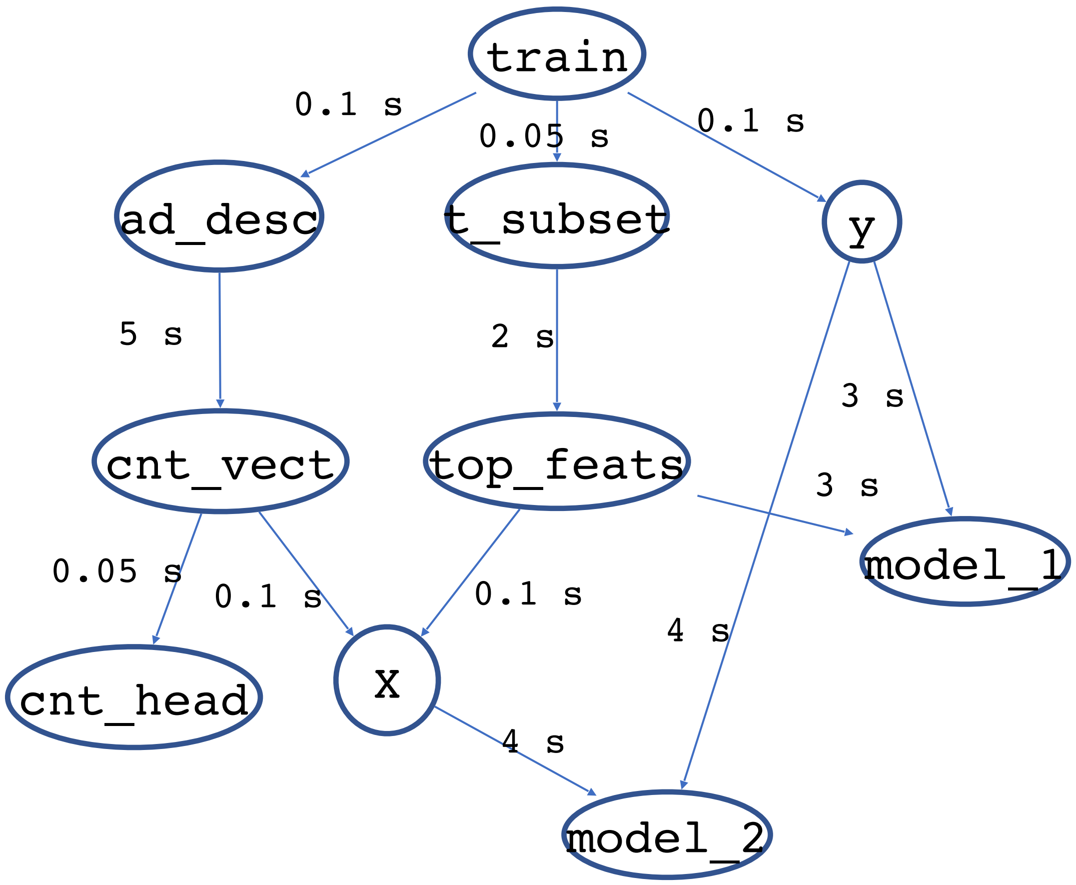


DAG Annotation



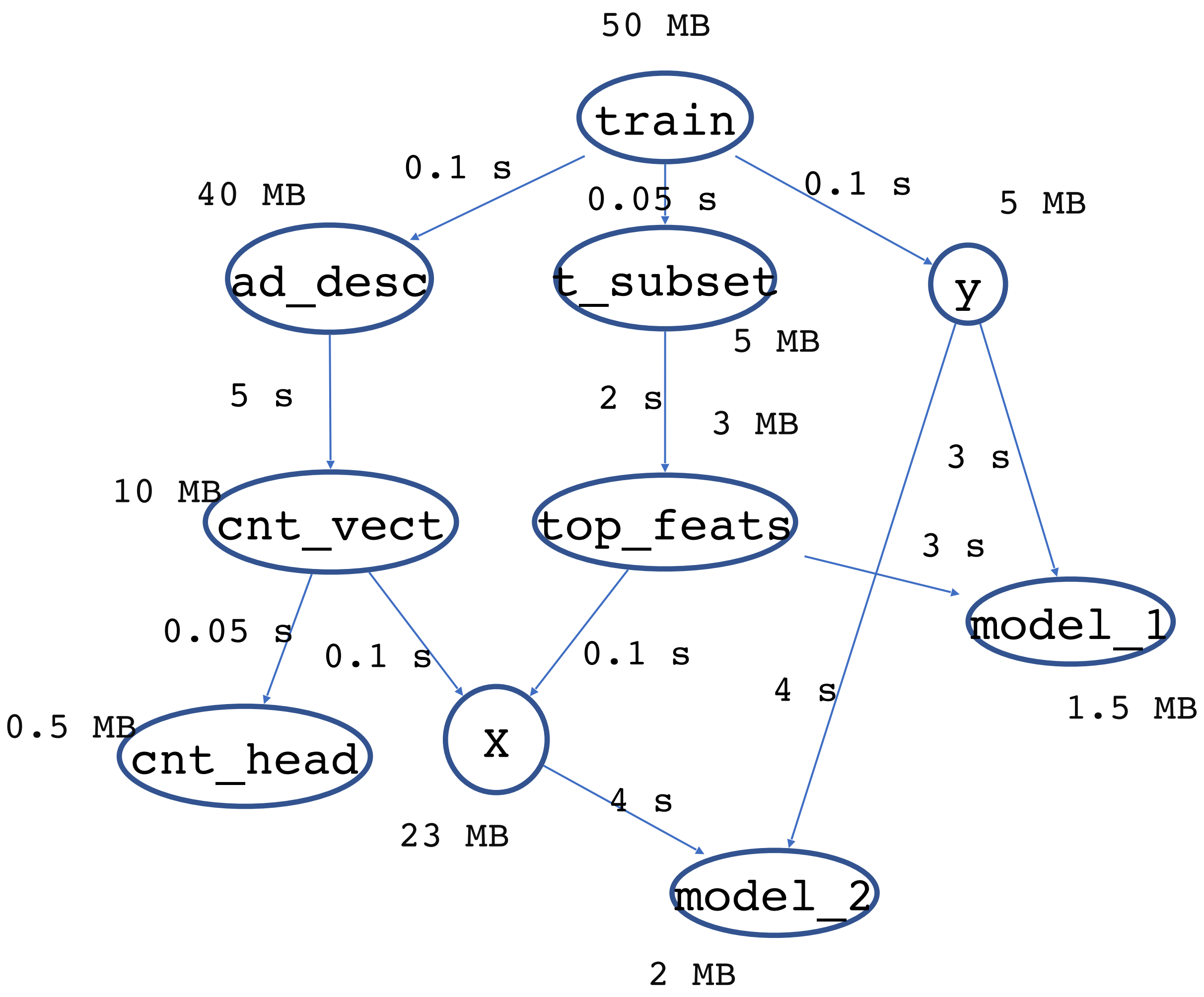
DAG Annotation

- Edge run-time



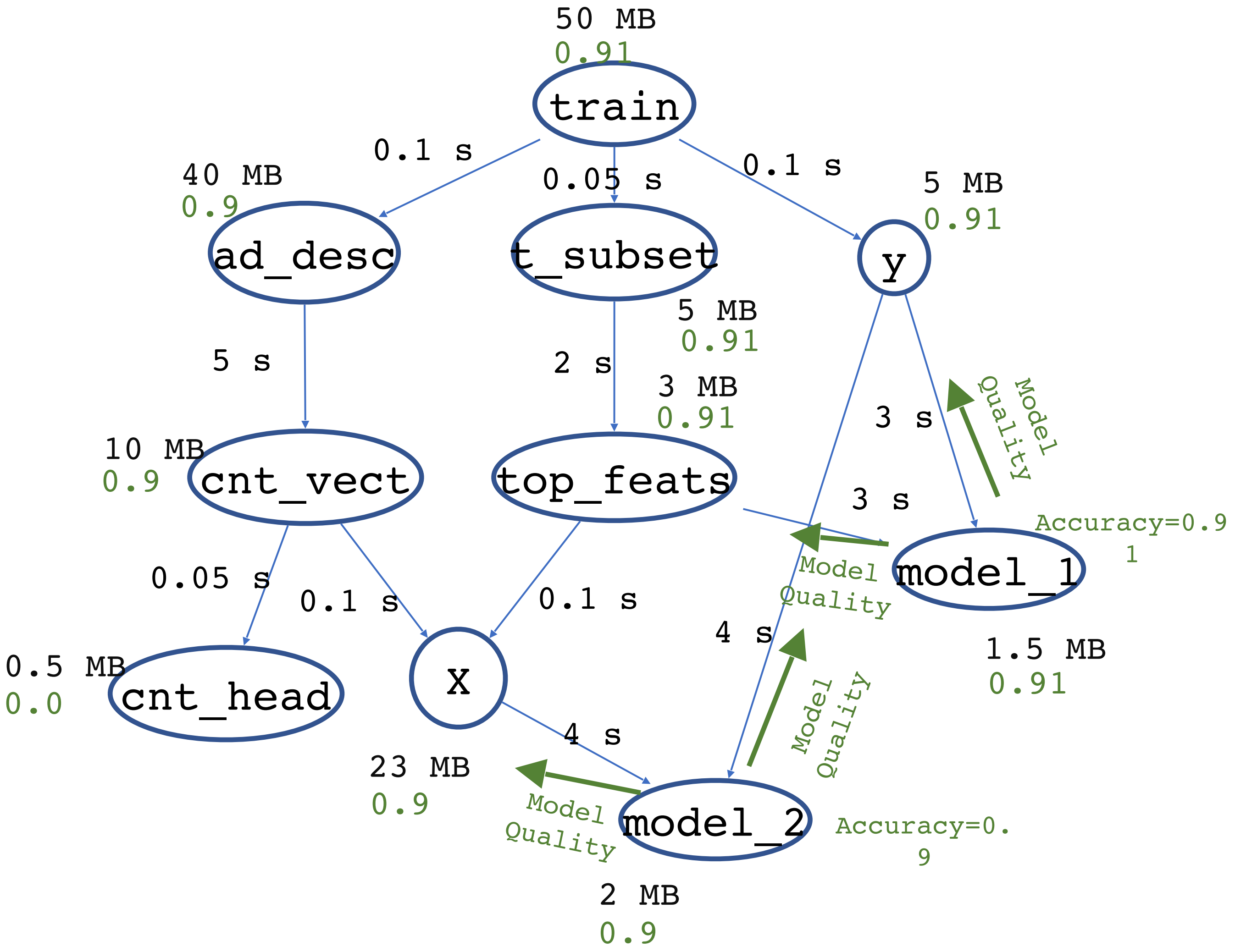
DAG Annotation

- Edge run-time
- Vertex size

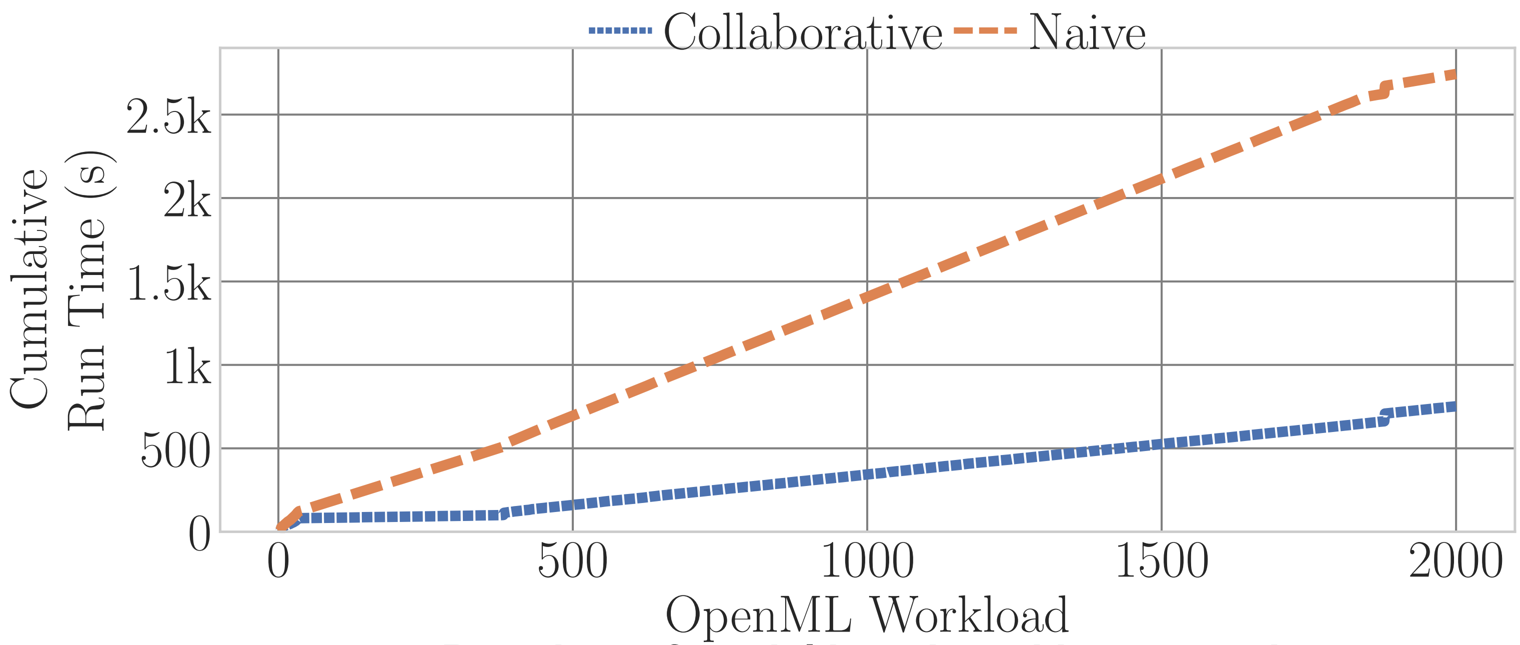


DAG Annotation

- Edge run-time
- Vertex size
- Vertex potential
    - Quality of the best reachable model



Model Materialization and Warmstarting

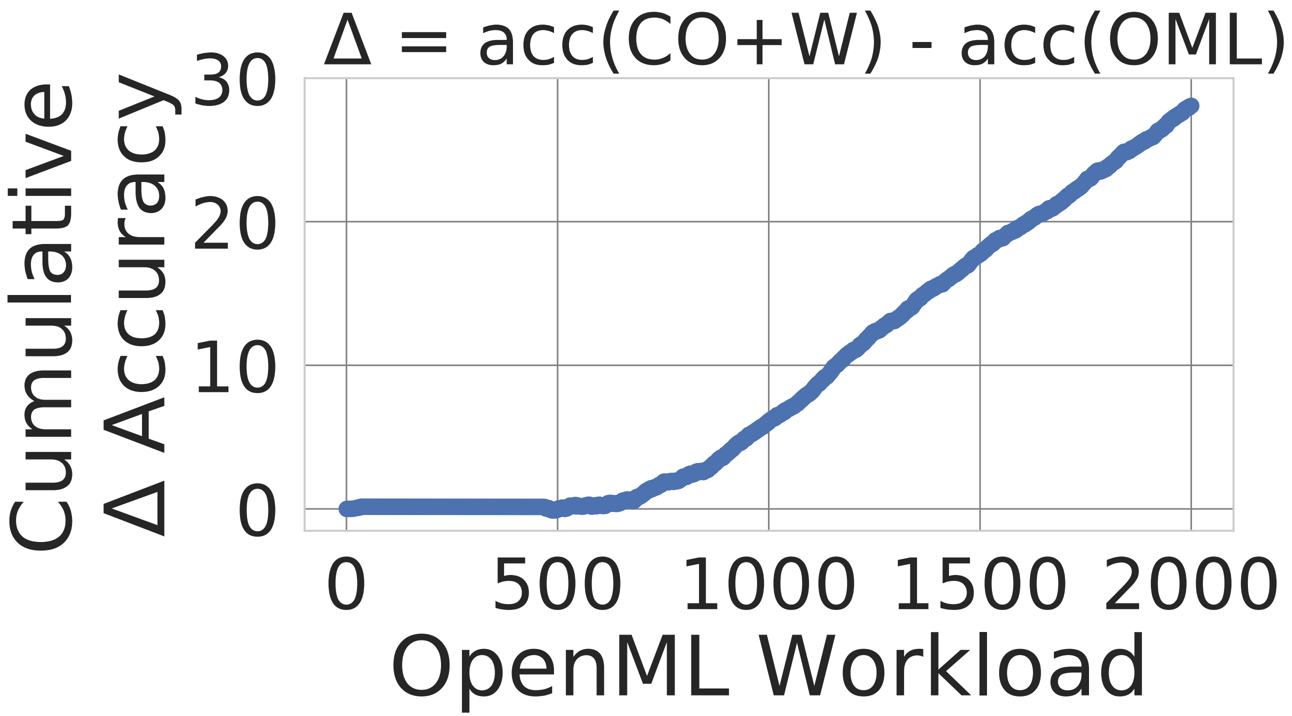


Run-time of model benchmarking scenario

German Research Center for Artificial Intelligence

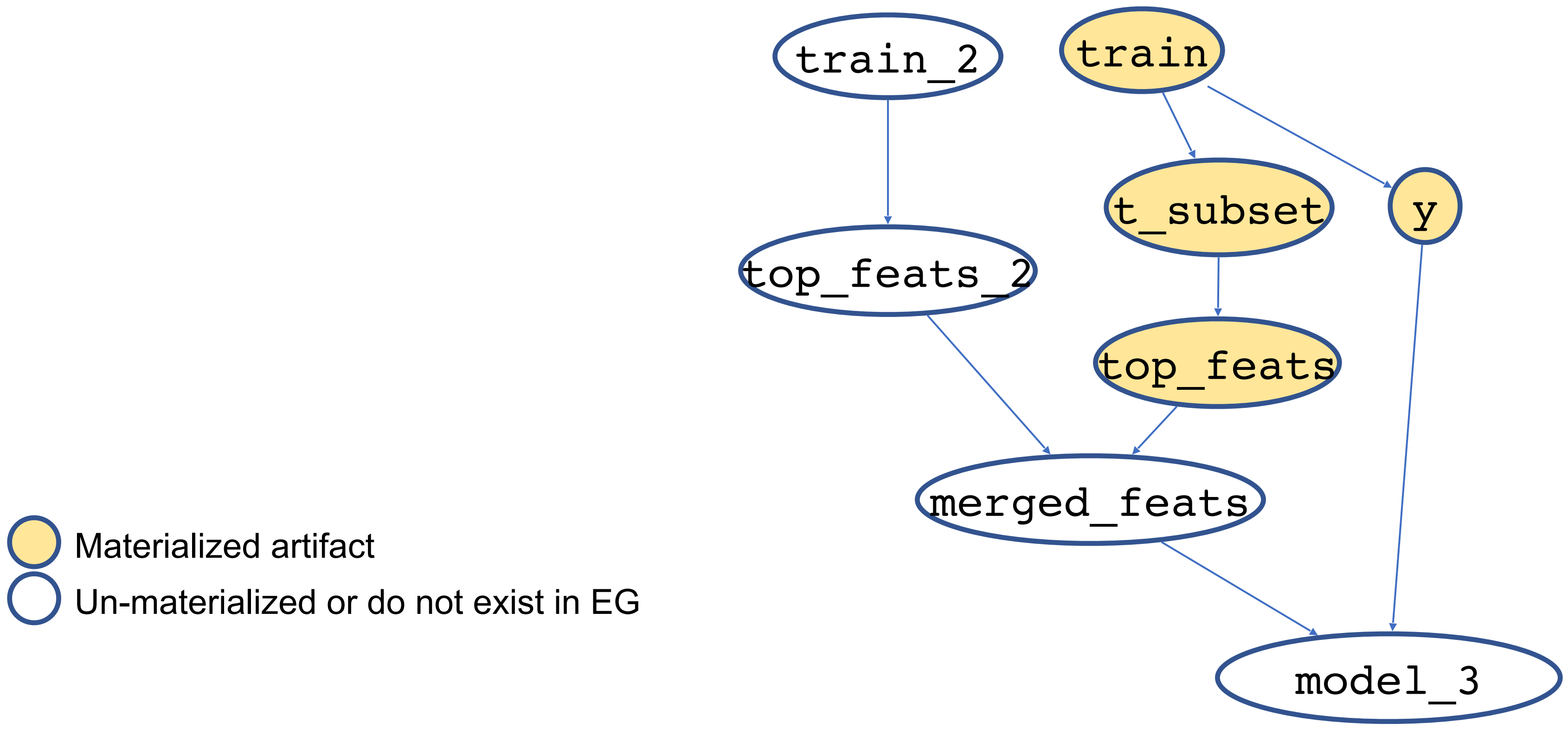
Technische Universität Berlin

Impact of Model Warmstarting on Accuracy



Reuse Algorithm

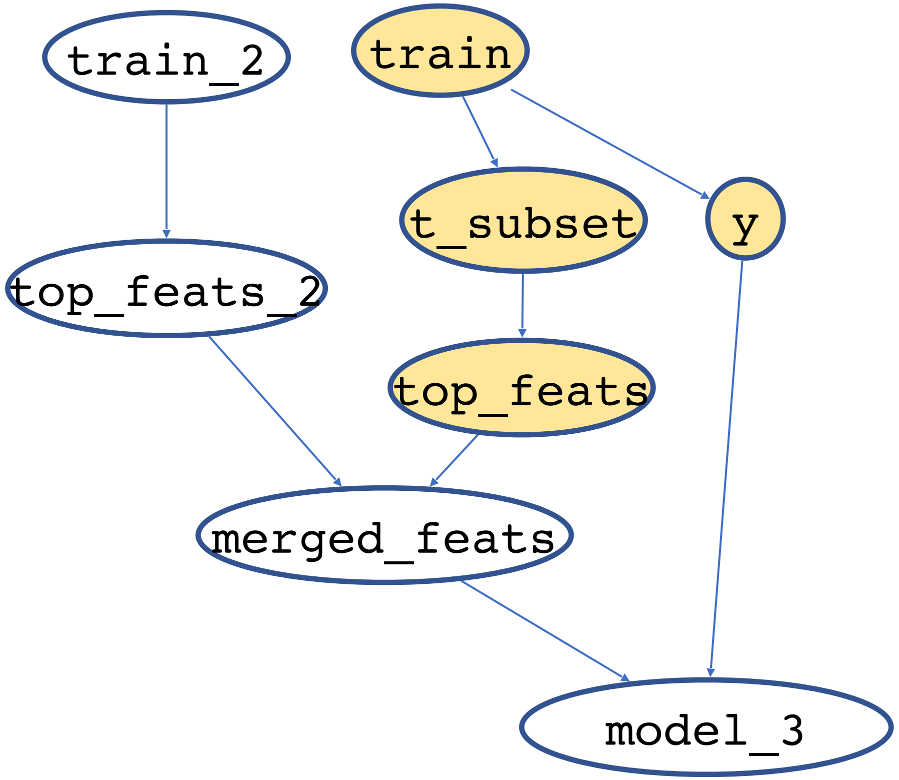
A linear-time algorithm to compute optimal execution plan with a ***forward and backward pass*** on the workload DAG



Reuse Algorithm (Forward-pass)

Forward-pass

1. Traverse from the source
2. Accumulate run-times
3. For every materialized node compare the **accumulated run-time** with the **load-time**



Materialized artifact

Un-materialized or do not exist in EG

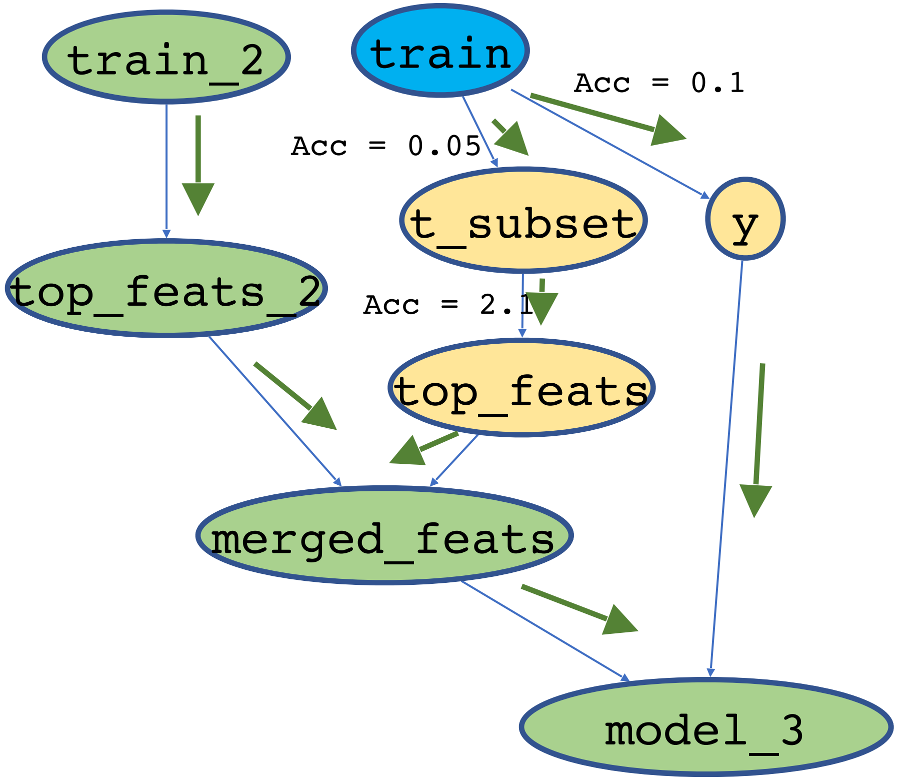
Artifacts to execute

Artifacts to load

Reuse Algorithm (Forward-pass)

Forward-pass

1. Traverse from the source
2. Accumulate run-times
3. For every materialized node compare the **accumulated run-time** with the **load-time**



Materialized artifact

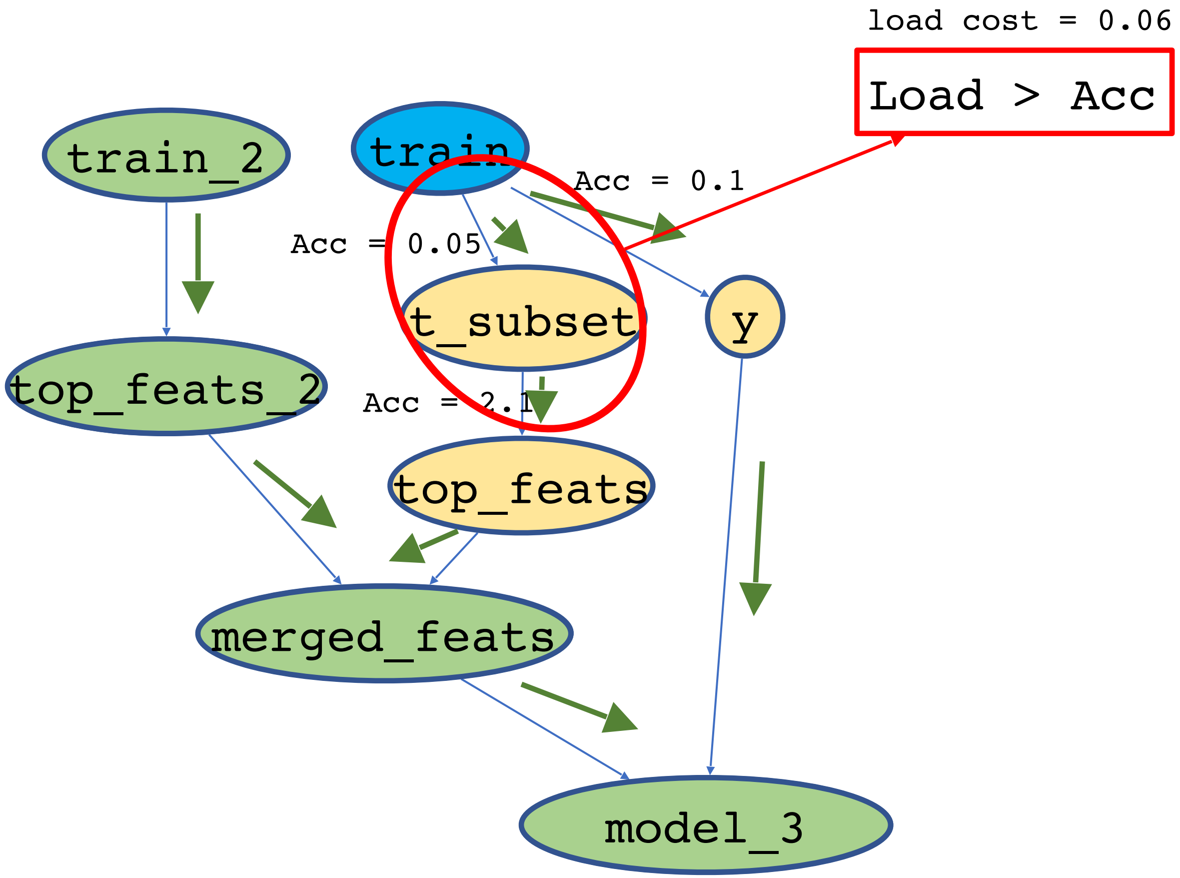
Un-materialized or do not exist in EG

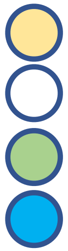
Artifacts to execute

Artifacts to load

Reuse Algorithm (Forward-pass)

Forward-pass
1. Traverse from the source 2. Accumulate run-times 3. For every materialized node compare the **accumulated run-time** with the **load-time**

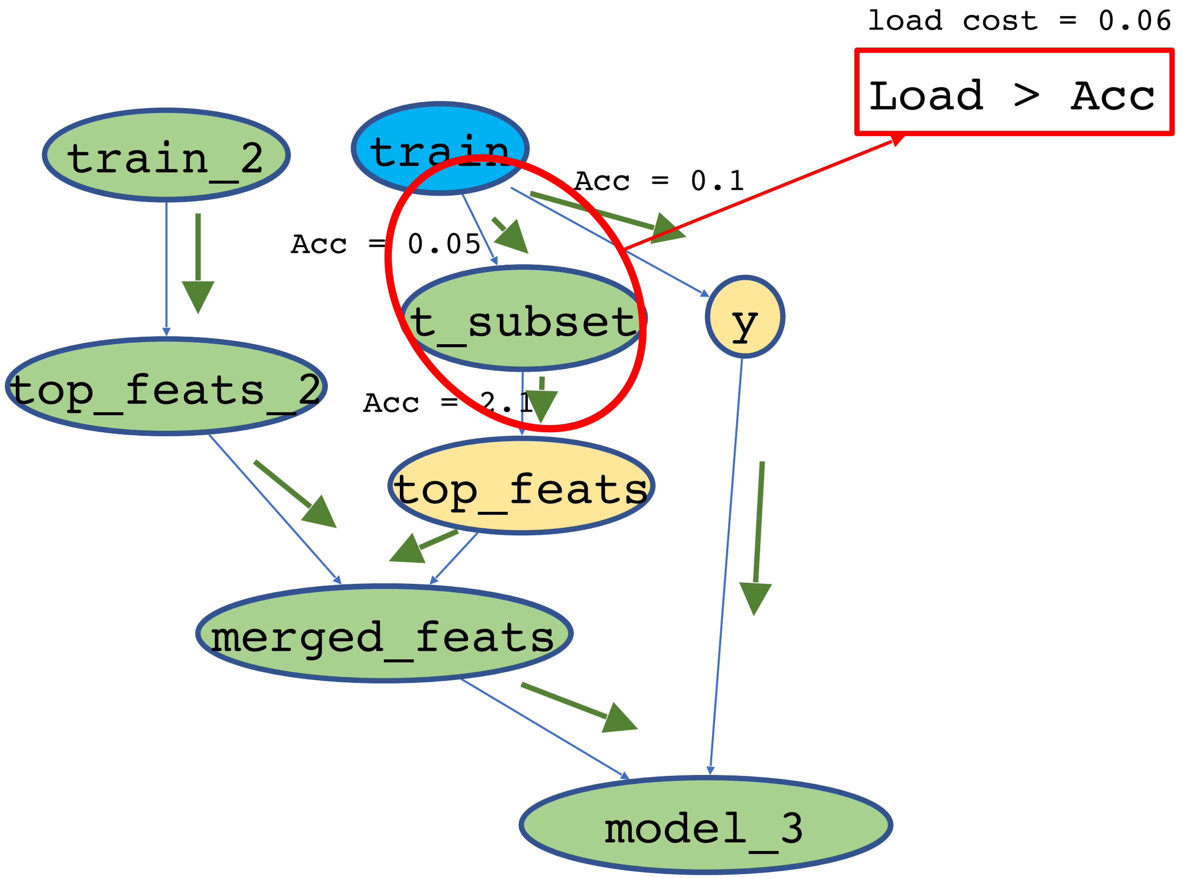




German Research Center for Artificial Intelligence
Technische Universität Berlin

Reuse Algorithm (Forward-pass)

Forward-pass
1. Traverse from the source 2. Accumulate run-times 3. For every materialized node compare the **accumulated run-time** with the **load-time**

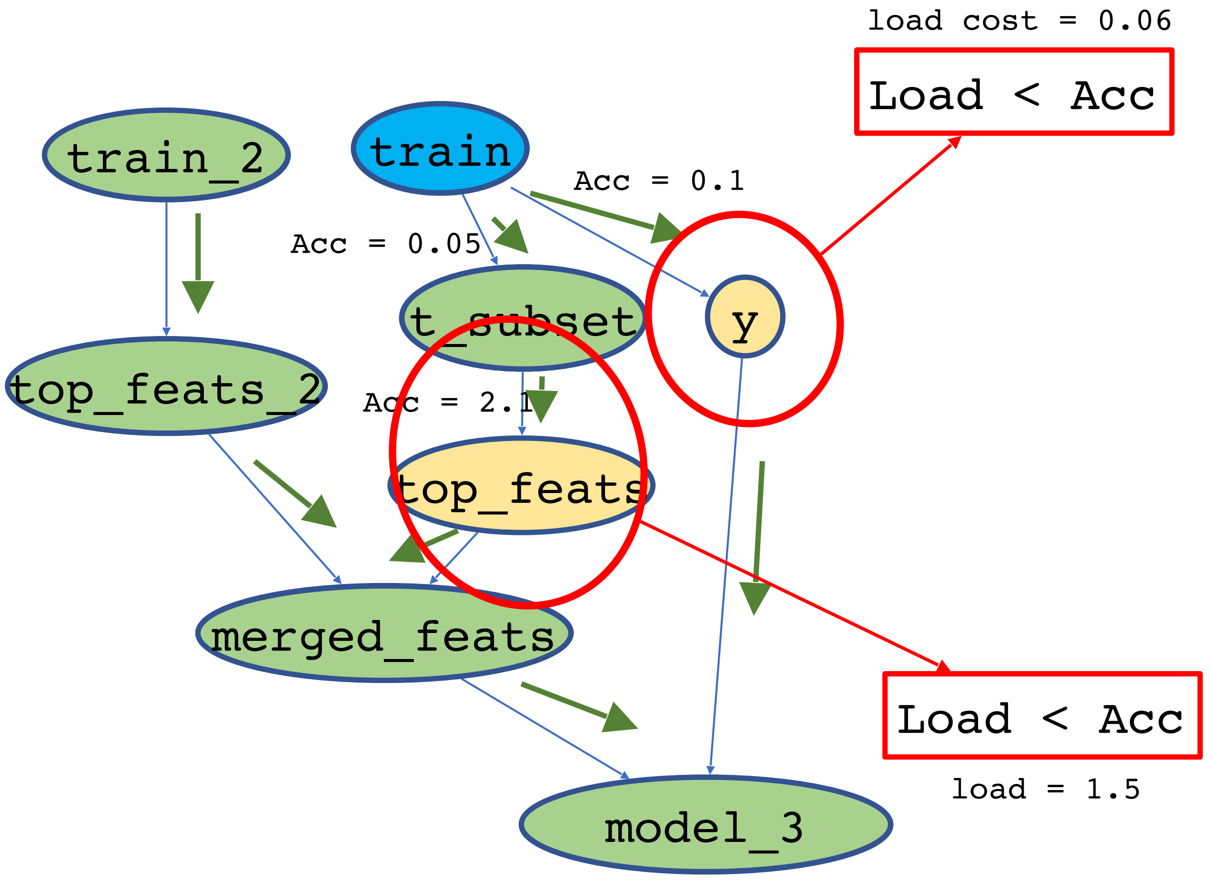


Materialized artifact
Un-materialized or do not exist in EG
Artifacts to execute
Artifacts to load

Reuse Algorithm (Forward-pass)

Forward-pass

1. Traverse from the source
2. Accumulate run-times
3. For every materialized node compare the **accumulated run-time** with the **load-time**

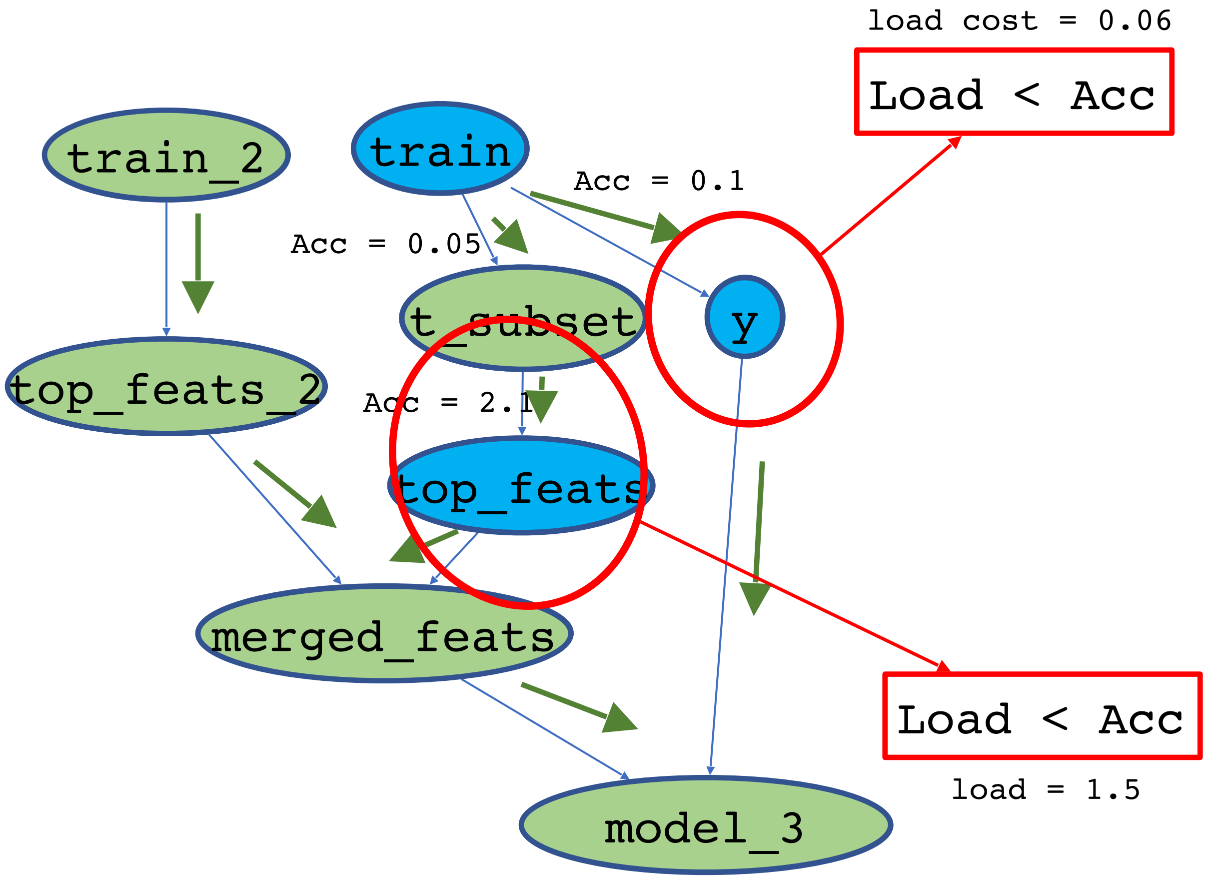


Materialized artifact
Un-materialized or do not exist in EG
Artifacts to execute
Artifacts to load

German Research Center for Artificial Intelligence
Technische Universität Berlin

Reuse Algorithm (Forward-pass)

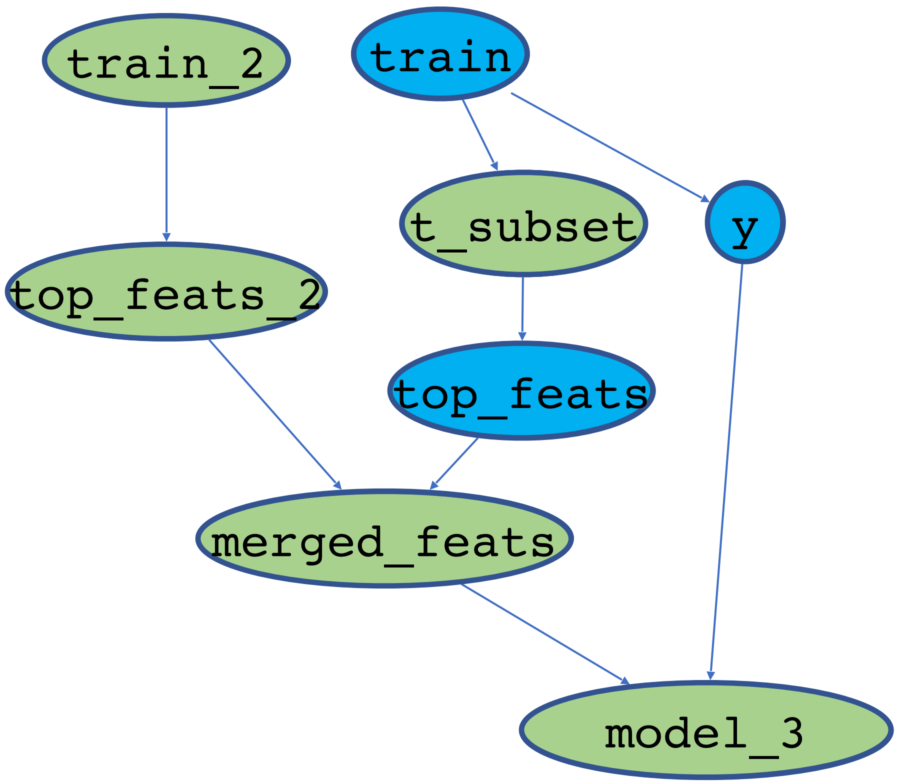
Forward-pass
1. Traverse from the source
2. Accumulate run-times
3. For every materialized node compare the **accumulated run-time** with the **load-time**



- Materialized artifact
- Un-materialized or do not exist in EG
- Artifacts to execute
- Artifacts to load

Reuse Algorithm (Forward-pass)

Forward-pass
1. Traverse from the source 2. Accumulate run-times 3. For every materialized node compare the **accumulated run-time** with the **load-time**

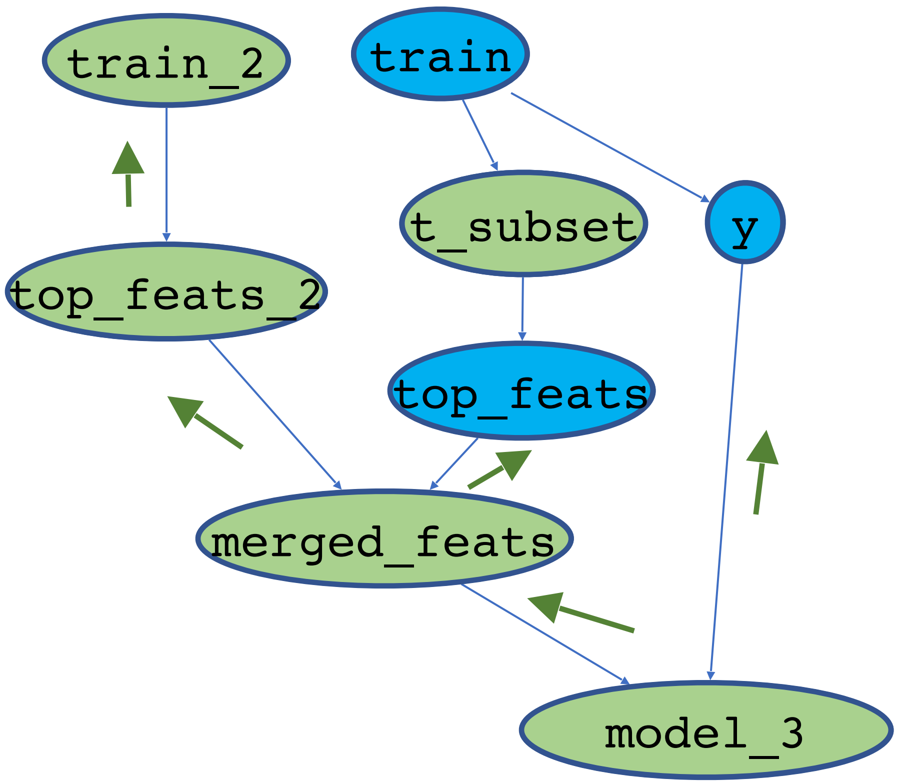


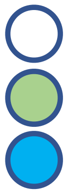
Materialized artifact
Un-materialized or do not exist in EG
Artifacts to execute
Artifacts to load

German Research Center for Artificial Intelligence
Technische Universität Berlin

Reuse Algorithm (Backward-pass)

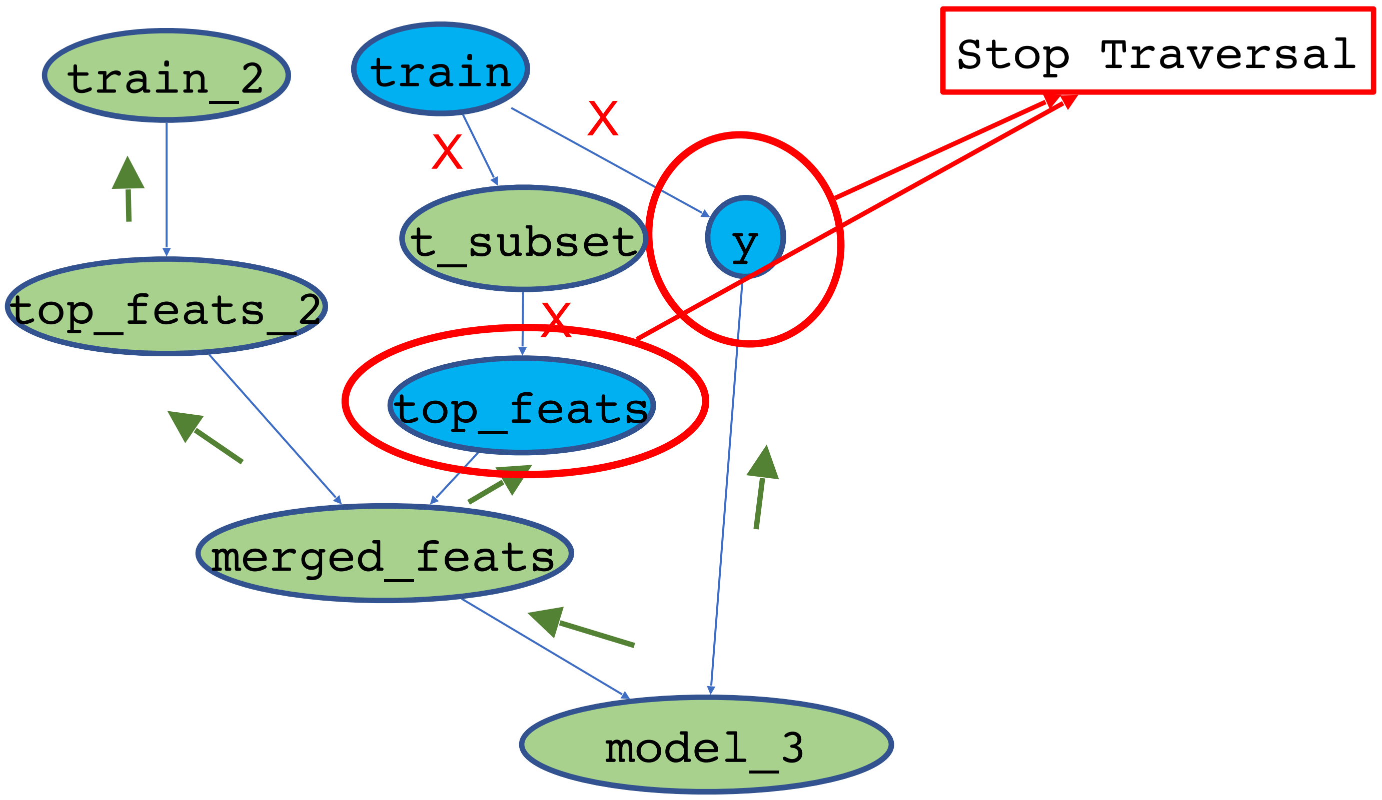
Backward-pass
1. Traverse backward from the terminal 2. For every **loaded** artifact, stop the traversal of its parents 3. **Prune** artifacts that are not visited





Reuse Algorithm (Backward-pass)

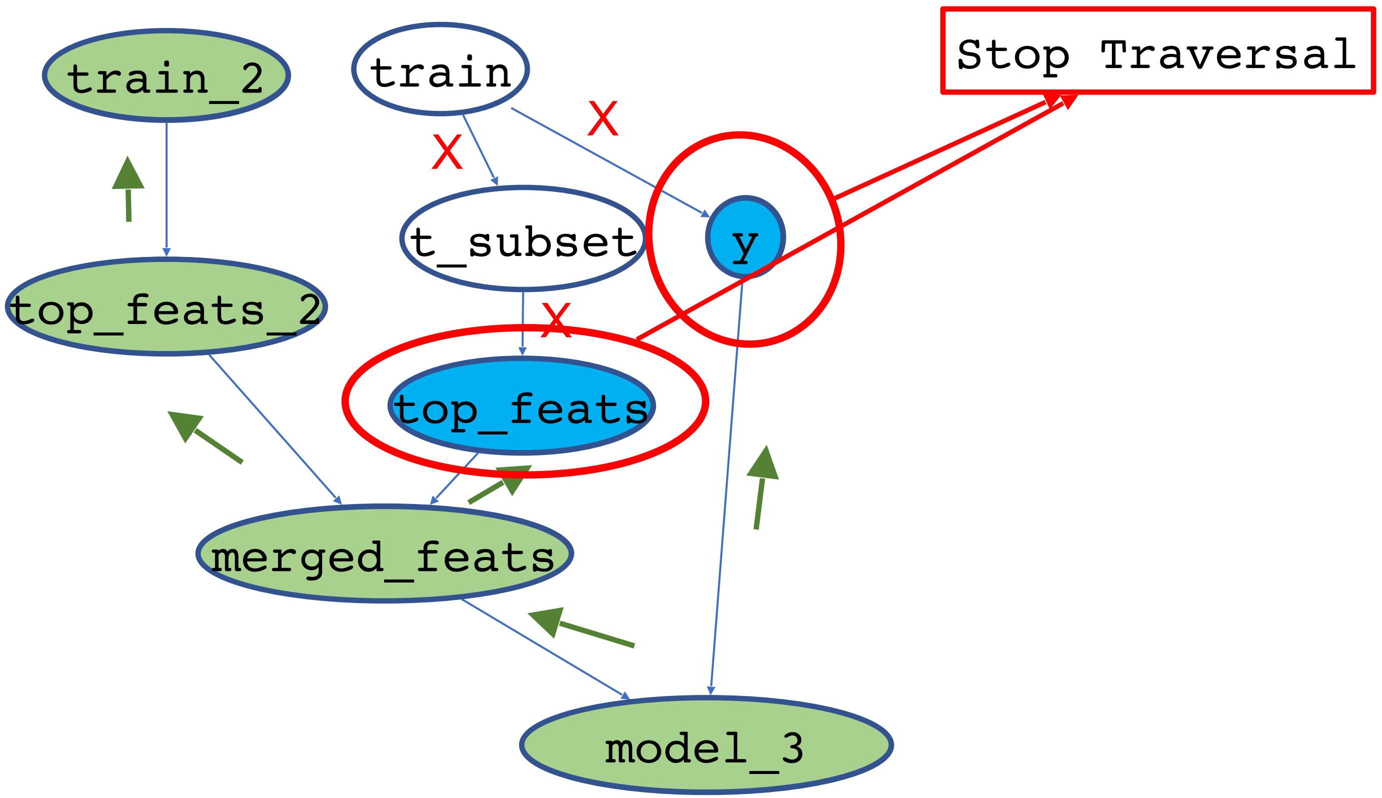
Backward-pass
1. Traverse backward from the terminal 2. For every **loaded** artifact, stop the traversal of its parents 3. **Prune** artifacts that are not visited

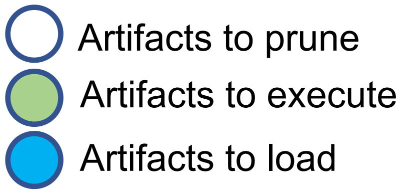


Artifacts to prune
Artifacts to execute
Artifacts to load

Reuse Algorithm (Backward-pass)

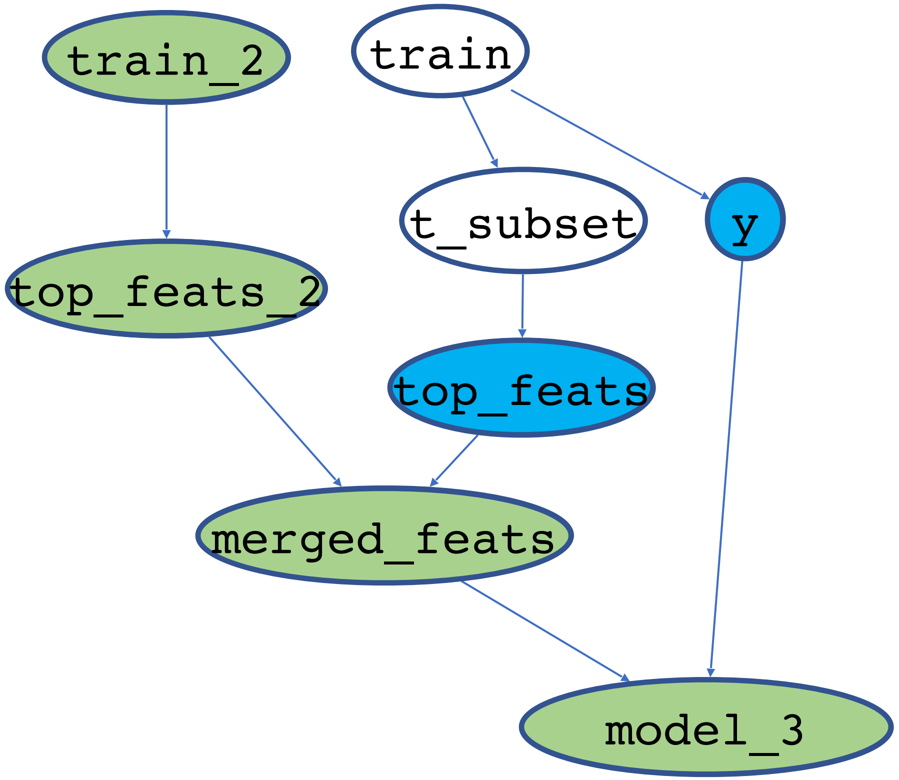
Backward-pass
1. Traverse backward from the terminal 2. For every **loaded** artifact, stop the traversal of its parents 3. **Prune** artifacts that are not visited





Reuse Algorithm (Backward-pass)

Backward-pass
1. Traverse backward from the terminal 2. For every **loaded** artifact, stop the traversal of its parents **3. Prune** artifacts that are not visited

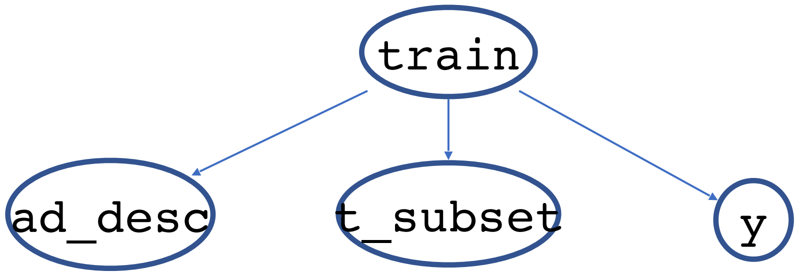


Artifacts to prune
Artifacts to execute
Artifacts to load

Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts
- Apply column deduplication strategy

```
train = pd.read_csv('train.csv') #
[ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc‘]
t_subset = train[['ts','u_id','price‘]]
y = train['y']
```

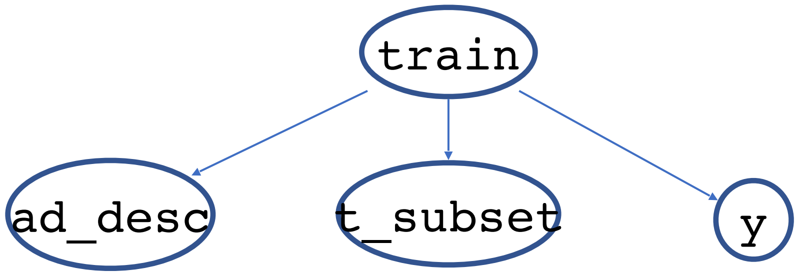


German Research Center for Artificial Intelligence
Technische Universität Berlin

Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts
- Apply column deduplication strategy

```
train = pd.read_csv('train.csv') #
[ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc‘]
t_subset = train[['ts','u_id','price‘]]
y = train['y']
```

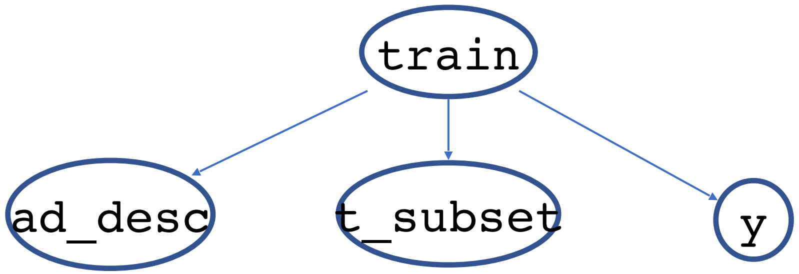


Storage-aware Materialization
while budget is not exhausted:
1. run materialization algorithm
2. deduplicate the materialized artifacts
3. update the size of unmaterialized artifact

Storage-aware Materialization

- Many **duplicated** columns in intermediate data artifacts
- Apply column deduplication strategy

```
train = pd.read_csv('train.csv') #
[ad_desc,ts,u_id,price,y]
ad_desc = train['ad_desc‘]
t_subset = train[['ts','u_id','price‘]]
y = train['y']
```



Storage-aware Materialization
while budget is not exhausted: 1. run materialization algorithm 2. deduplicate the materialized artifacts 3. update the size of unmaterialized

Improves Storage utilization and Run-time

German Research Center for Artificial Intelligence

Technische Universität Berlin