GRACEFUL – A Learned Cost Estimator For UDFs

Johannes Wehrstein *, Tiemo Bang †, Roman Heinrich *‡, Carsten Binnig *‡

* Technical University of Darmstadt, † Microsoft – Gray Systems Lab, ‡ DFKI

41st IEEE International Conference on Data Engineering
— HONG KONG SAR, CHINA | MAY 19 – 23, 2025 —

TECHNISCHE UNIVERSITÄT DARMSTADT

Imagine you are a data-scientist…

… and want to retrieve all customers with a churn risk larger than 80%.

Setup:

- the data is in a database
- your churn metric is written in Python

```python
def churn(user) -> bool:
    # Base probability
    churn_probability = 0.2

    # Adjust probability based on inputs
    if tenure_months < 6:
        churn_probability += 0.2
    if monthly_spend < 30:
        churn_probability += 0.1
    if support_calls > 3:
        churn_probability += 0.15
    if tenure_months > 24:
        churn_probability -= 0.1

    churn_probability = min(max(churn_probability,
        0), 1)  # Clamp to [0, 1]

    return random.random() < churn_probability
```

Solution: Use UDFs to Execute Python code in the DBMS

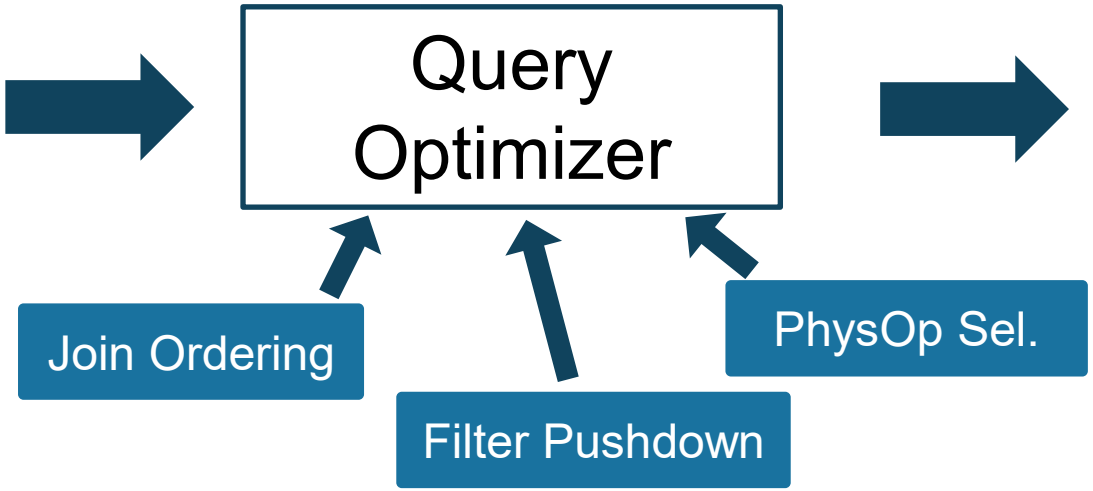
*SELECT * FROM users as u WHERE* ***churn(u)****>0.8;*

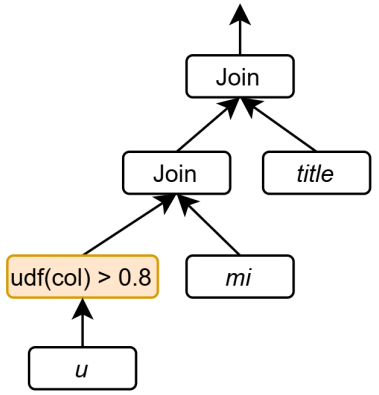
UDF during QO

Expectation:

```
SELECT COUNT( *)
FROM users AS u JOIN …
WHERE churn(u) > 0.8;
```

Query with **UDF**





Optimized Query Plan

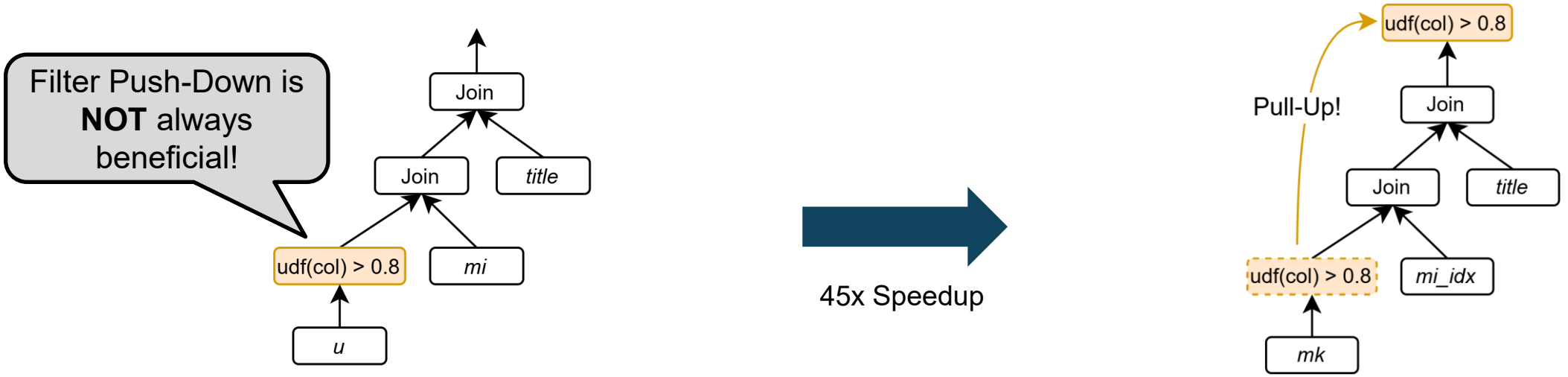
Unfortunately, this is not the reality!

Many things go wrong during QO for UDFs

Example: Filter Push-Down

Query with **UDF:**

SELECT COUNT(∗) **FROM** users AS u **JOIN … WHERE** *churn(u)* > 0.8;



Filter **Push-Down**

21.86 s

Filter **Pull-Up**

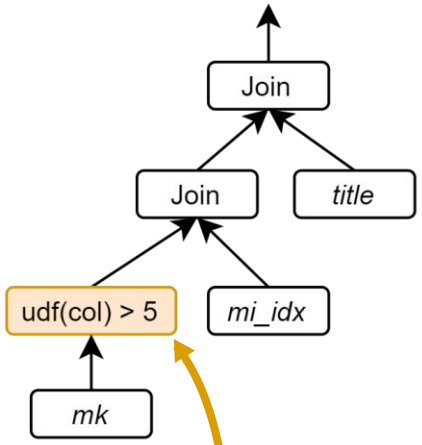
0.48 s

How to decide when to apply pull-up?

A Cost Estimator for UDFs is needed

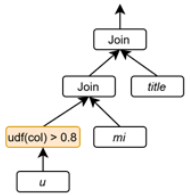
Cost Estimation for UDFs is a hard problem

1. **Undecidable** problem in general: c.f. halting problem
2. Every UDF is **different:** complexity / length / operators
3. **Different runtimes** for tuples: if/else conditions
4. **No information** on **Cardinalities** inside and above UDF: output of the UDF and branching is unknown




```
1:  def func (x, y) :
2:      if x < 20:
3:          z = x ** 2
4:      else :
5:          for i in range(100):
6:              z = math.pow(math.sqrt(y), i) + z
7:      return z
```

Cost Estimation for UDFs is challenging



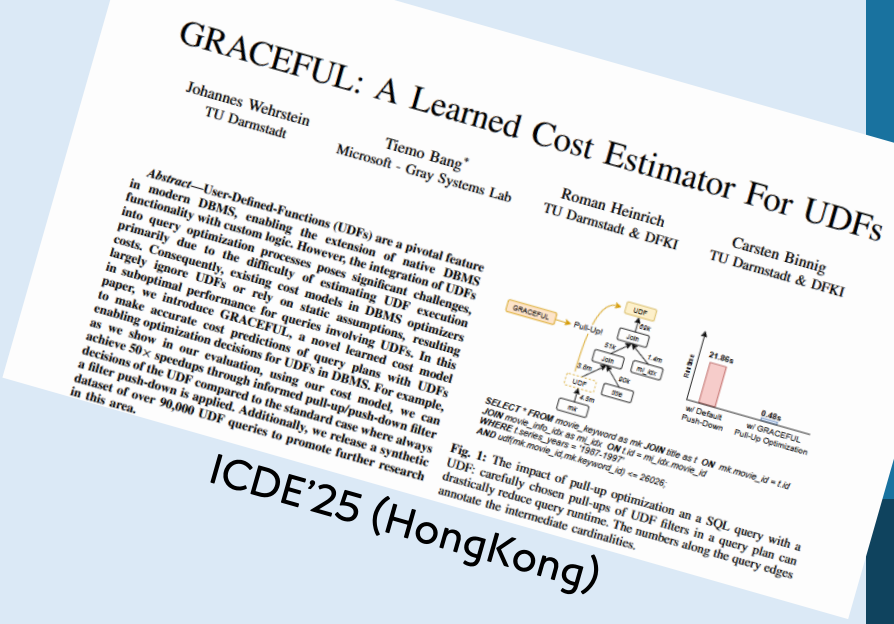

```
1:  def func (x, y) :
2:      if x < 20:
3:          z = x ** 2
            ...
7:      return z
```

Query Plan **UDF Code**

GRACEFUL

Learned Cost-Model

1.4s



GRACEFUL: A Learned Cost Estimator For UDFs

Johannes Wehrstein, TU Darmstadt
Tiemo Bang*, Microsoft - Gray Systems Lab
Roman Heinrich, TU Darmstadt & DFKI
Carsten Binnig, TU Darmstadt & DFKI

ICDE'25 (HongKong)

GRACEFUL

A Learned Cost-Estimator For UDFs

GRAph-based **C**ost **E**stimator **F**or **U**ser defined **L**ogic

Key Ideas

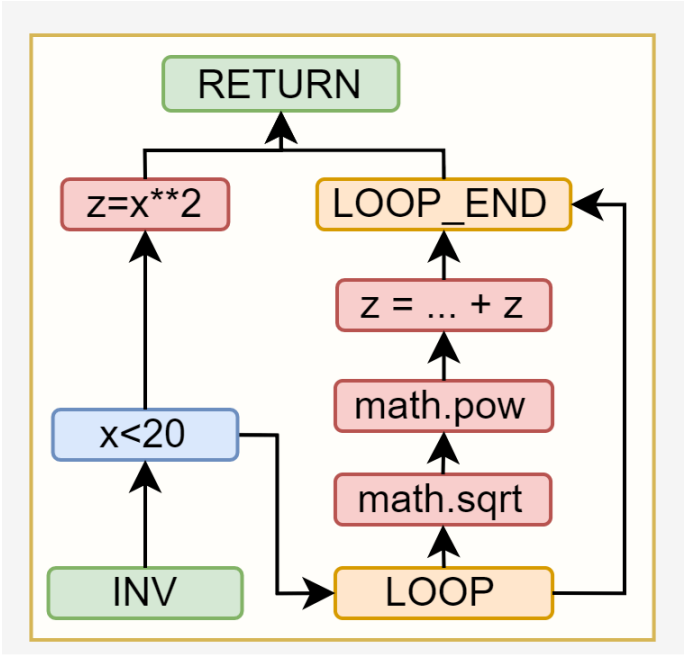
1. Representation as a **Graph**
2. **Transferable** Features
3. **Selectivity Estimation** inside UDF
4. Representing **UDF & Query Plan** together

#1 Transferable Representation of UDF as Graph

Split UDF into fine-granular operations
(Instead of representing as a black-box)

→ **representation as a graph**

1. Enables better understanding of the inner workings of the UDF
2. Allows Generalization to unseen code



Naïve representation as CFG is not enough:

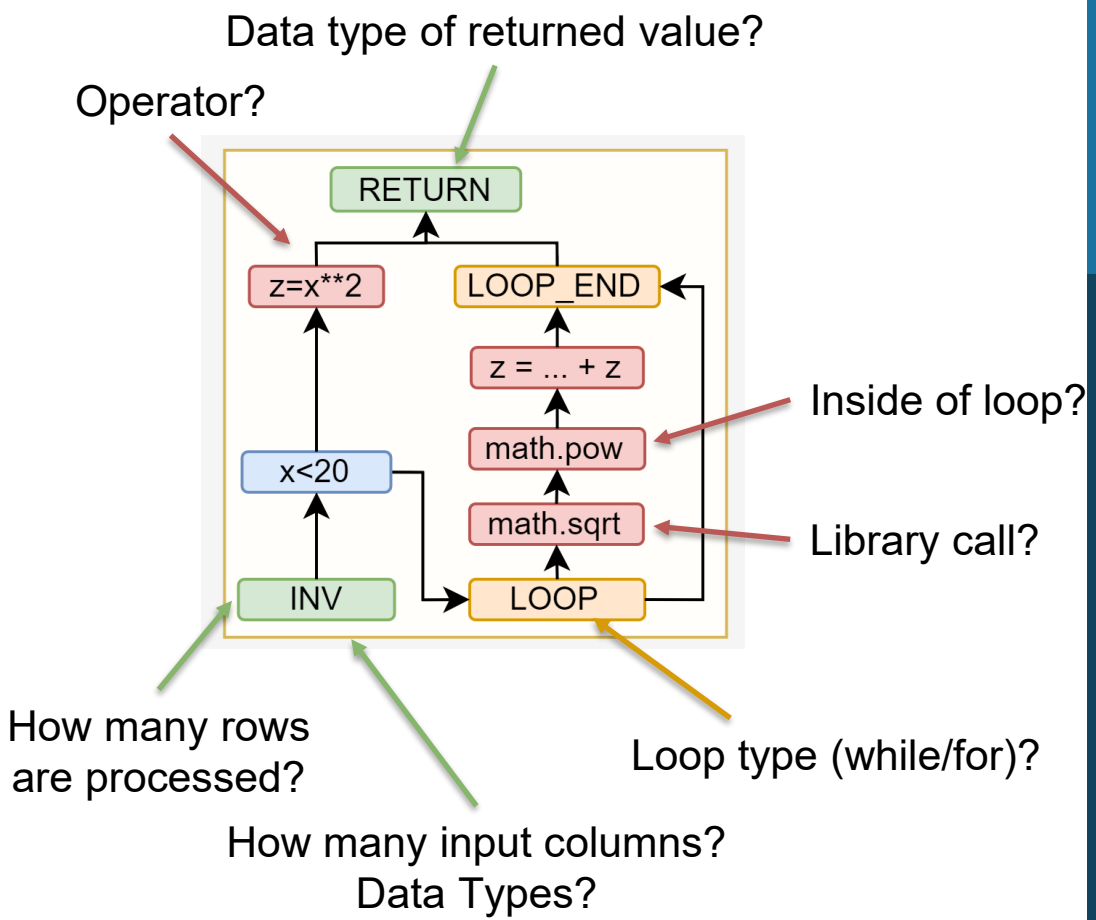
1. Collapsing of nodes (compact representation)
2. Additional edges (~residual connections)
   e.g. LOOP→ END_LOOP

#2 Transferable Featurization

Featurize abstract signature of UDF
(in contrast to featurizing code – var names could change, …)

Features:

1. Information related to computational complexity
2. On how many rows executed

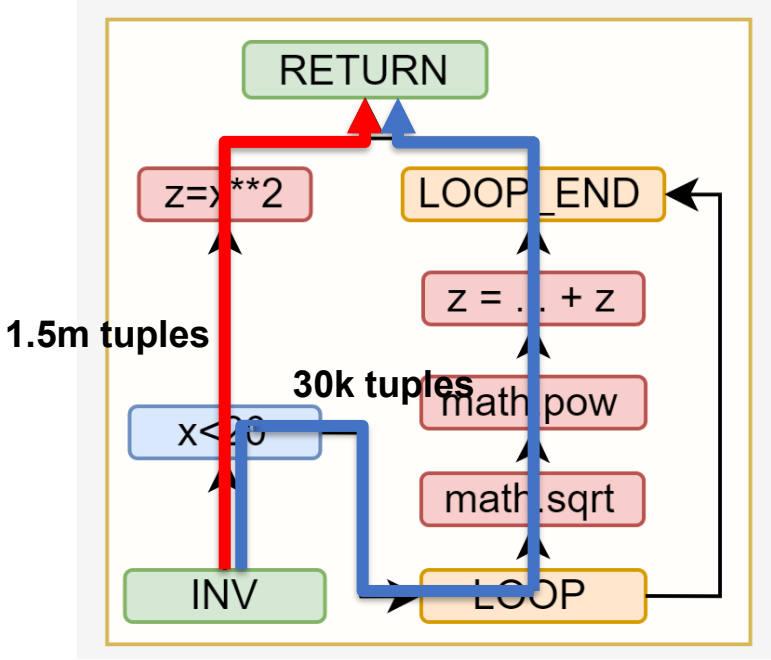


Allows generalization to unseen UDFs

#3 Selectivities inside UDF

Different paths in UDF can have different runtimes

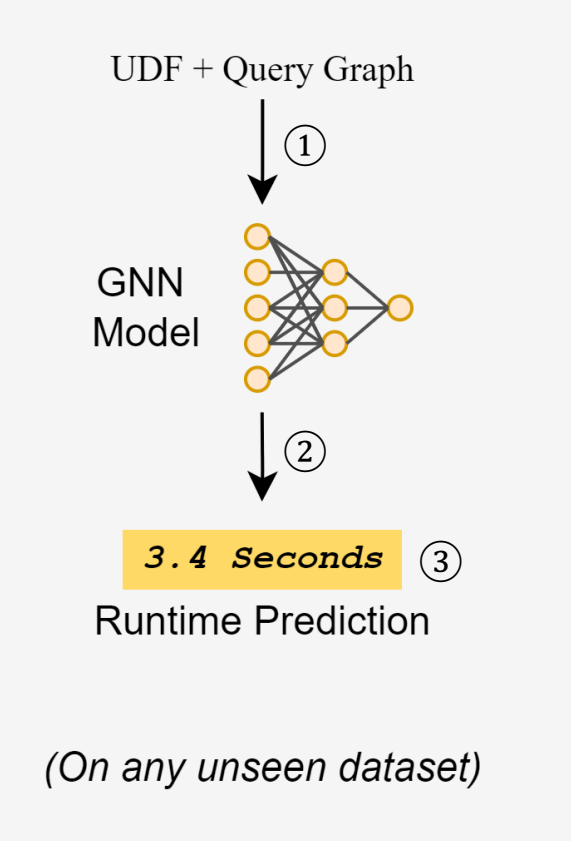
→ **Selectivities of IF/ELSE conditions important!**



Solution:

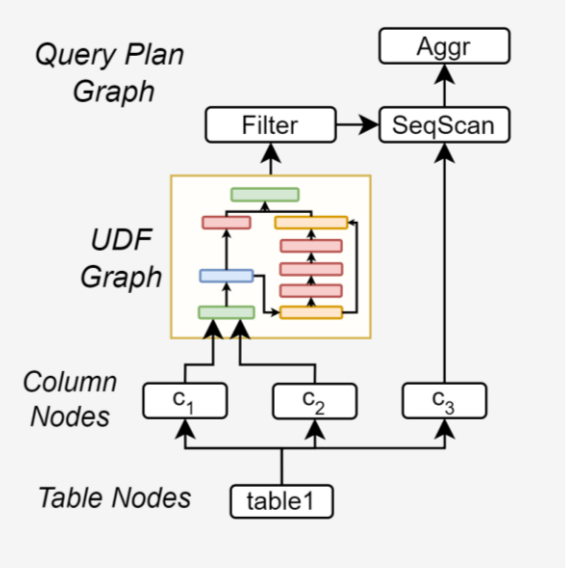
- Translate selectivity estimation problem into cardinality estimation problem
- Utilize cardinality estimator of DBMS

Run GNN Model



High level:

1. Feed unified graph structure to **Graph Neural Network**
   - Graph-MLP (on Heterograph)
   - Topological Message Passing
   - Readout at Root Node
2. Return a **unified embedding** of UDF & Query Plan
3. Predict Runtime with **Regression Model**



Synthetic Benchmark Generation

Workload Generator: synthetically generate UDFs & SQL queries

- Mimicking real-world UDFs based on Gupta et al.
- On 20 different databases

SQL Query:

```
SELECT COUNT( *)
FROM users AS u JOIN …
WHERE churn(u) > 0.8;
```

Any **SPJA** SQL-Query

UDF in **SELECT** and **WHERE**

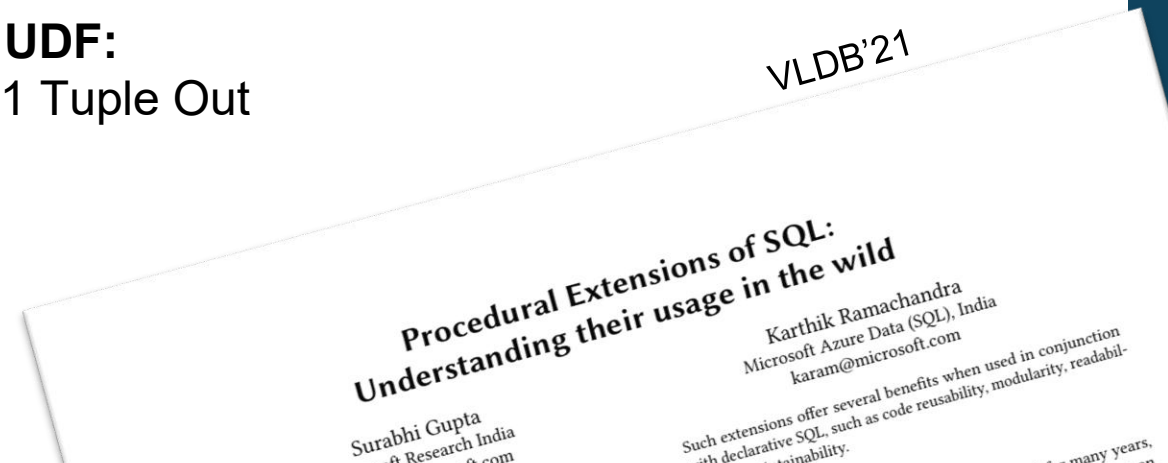
UDF:

```
1:  def func (x, y) :
2:      if x < 20:
3:          z = x ** 2
            ...
7:      return z
```

Scalar UDF:
1 Tuple in → 1 Tuple Out

Python UDF with:

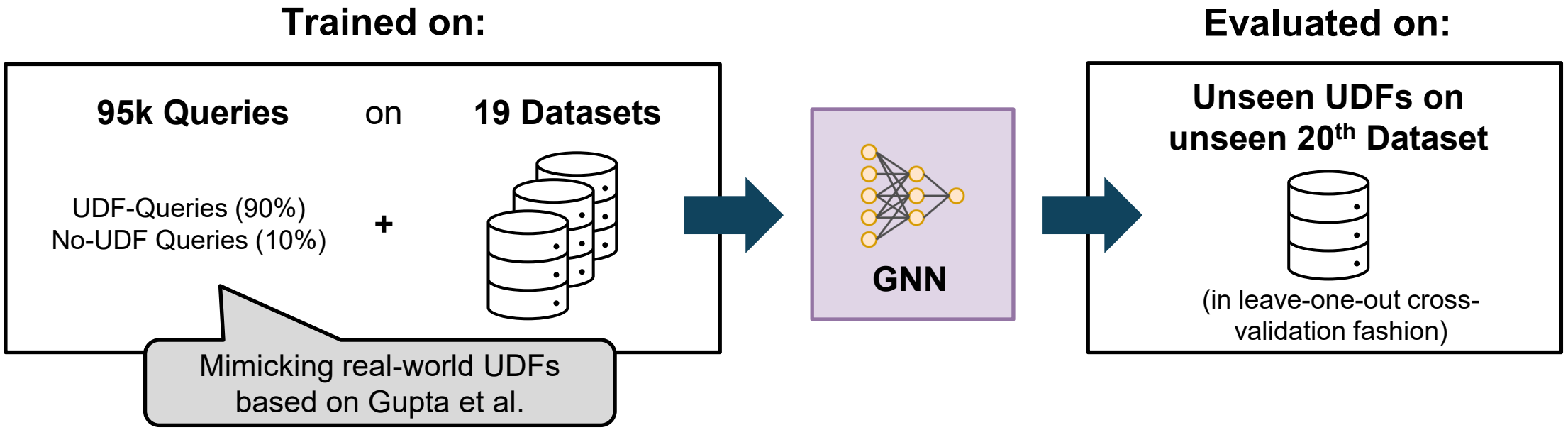
- Loops
- Branches
- Arithmetic/String Ops
- Library Calls



UDF-Benchmark & Code available on Github:
https://github.com/DataManagementLab/Graceful

Model Training

UDF-Benchmark & Code available on Github:
https://github.com/DataManagementLab/Graceful

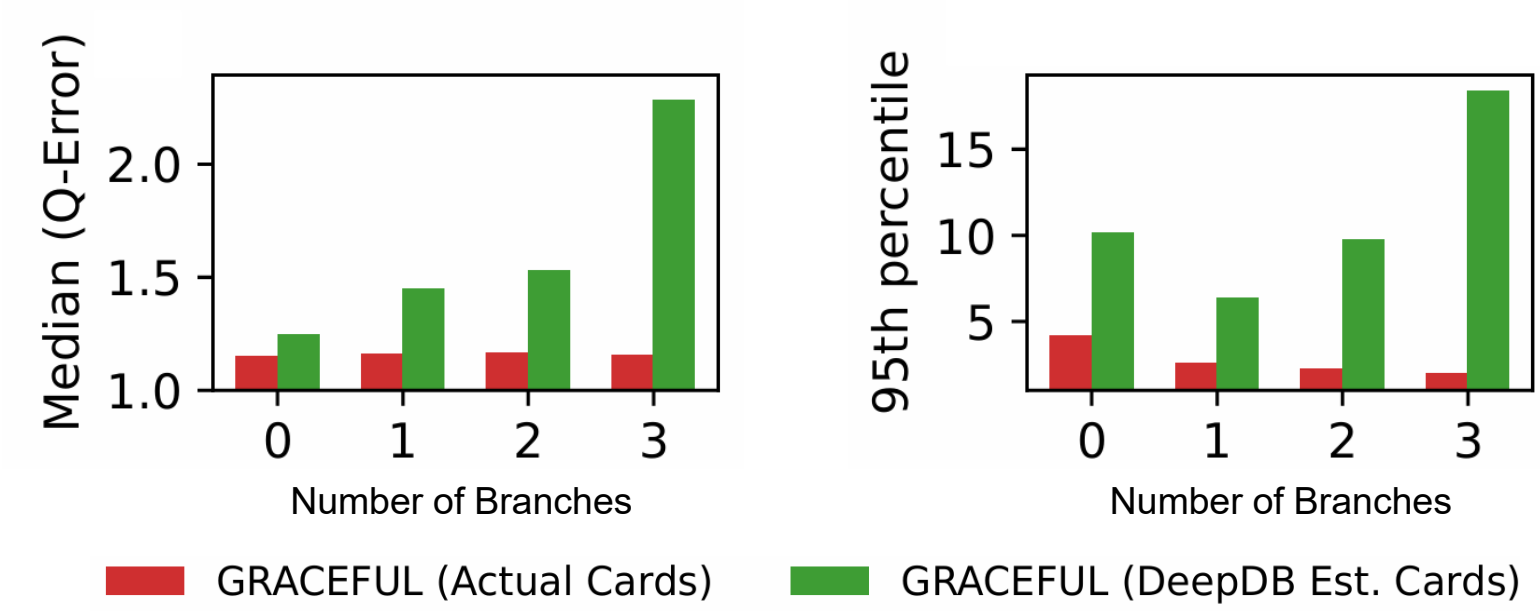


Evaluation of the model in a zero-shot fashion (unseen database, query & UDF)

Evaluation – Median Q-Error

Evaluated on unseen Dataset, UDF & SQL Queries

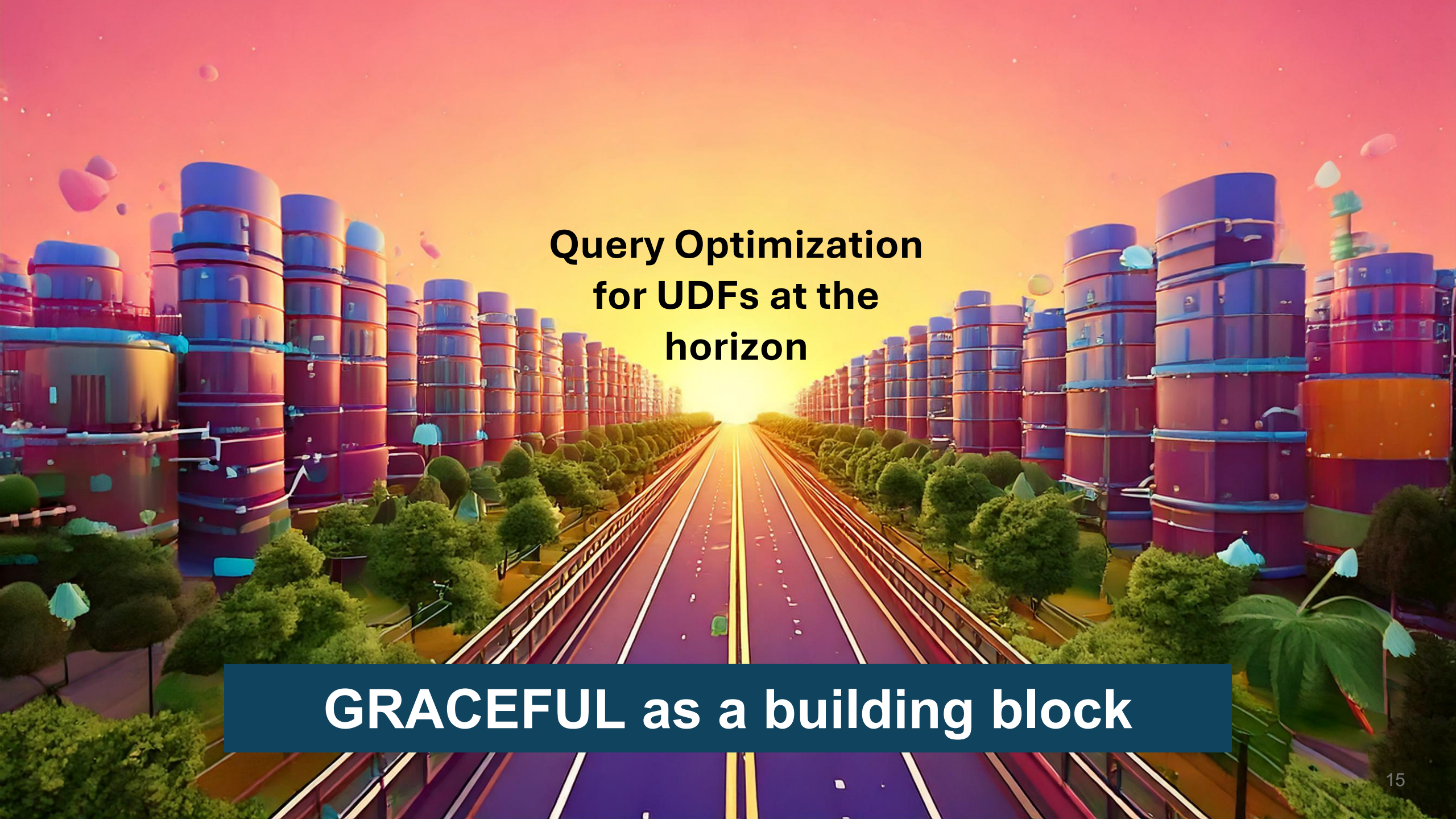
- Generalize across datasets (→ experiment in paper)
- Generalize across UDF complexity



Q-Error:

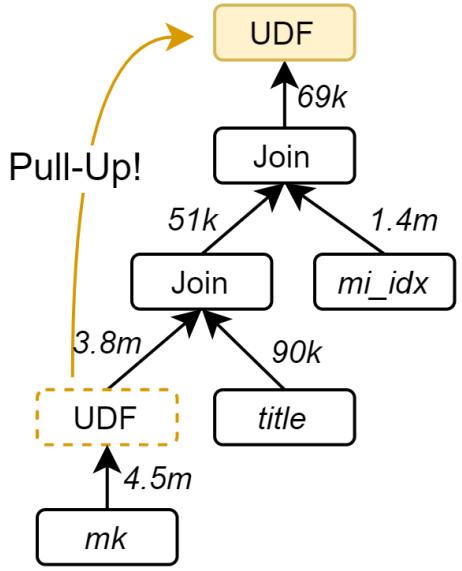
Relative Error Metric
(Lower is better, 1 is perfect)

Generalizes across datasets & UDF complexity


Query Optimization
for UDFs at the
horizon
GRACEFUL as a building block

Pull-Up / Push-Down Advisor

The placement of an UDF can make drastic differences (orders of magnitude speedups)



Example Transformation in UDF

Input Values	UDF Output
0.3	13
0.2	100
0.1	-10
…	…

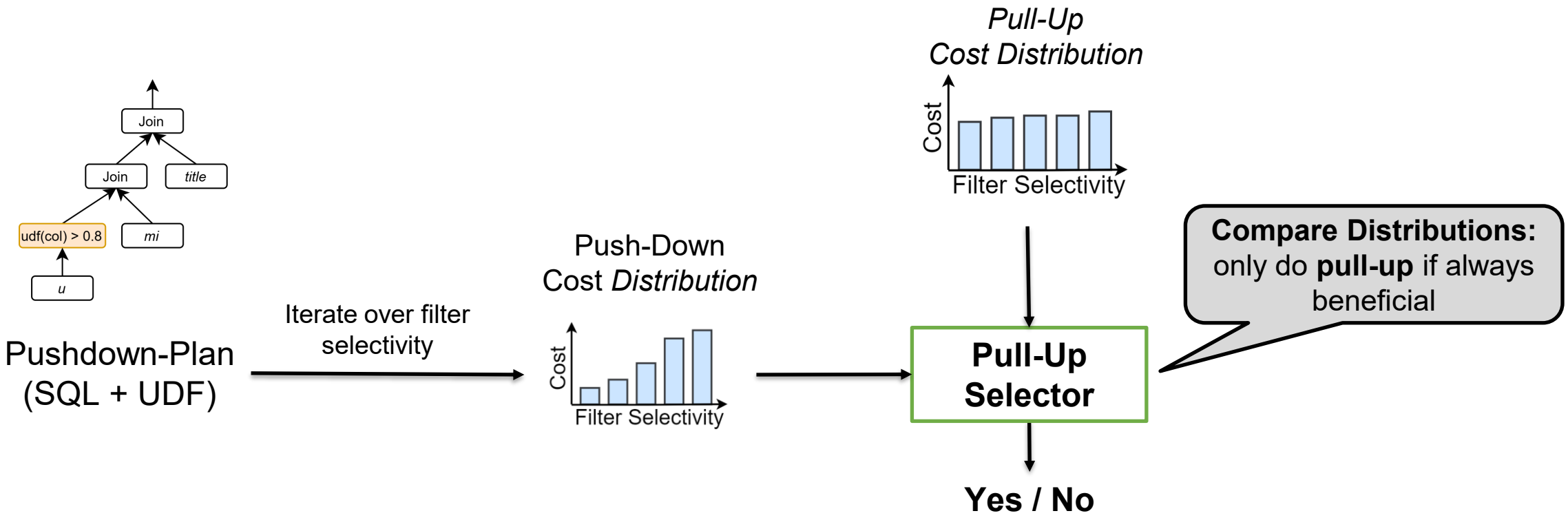
Transformation of Data in UDF unknown

→ **Cardinalities & cost of subsequent operators unknown** ❌

No idea of cost beyond UDF:
we have to work with uncertainty

Pull-Up / Push-Down Advisor

Goal: Decide Pull-Up / Push-Down **without cardinality information**



Regret optimization based on cost distribution

Pull-Up / Push-Down evaluation

Compare runtimes of pull-up/push-down plan selection

Further metrics in the paper

Selection Strategy	Total Runtime (hrs)	Total Speedup	False Positives
Optimal Pull-Up / Push-Down	3.082	1.643	-
GRACEFUL (Act. Card)	3.217	1.574	0.094
GRACEFUL	3.460	1.463	0.085
No Pull-Up	5.063	1	-

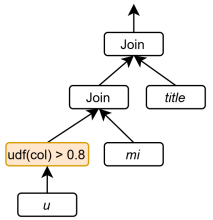
Default in DBMS

No cardinality information

Almost maximal speedups although very little information available

Overhead of our Optimizer: 3-3.5% of workload runtime (unoptimized system)

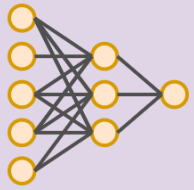
GRACEFUL




```
1:  def func (x, y) :
2:      if x < 20:
3:          z = x ** 2
            ...
7:      return z
```

Query Plan **UDF Code**

GRACEFUL



Learned Cost-Model

1.4s

Contributions:

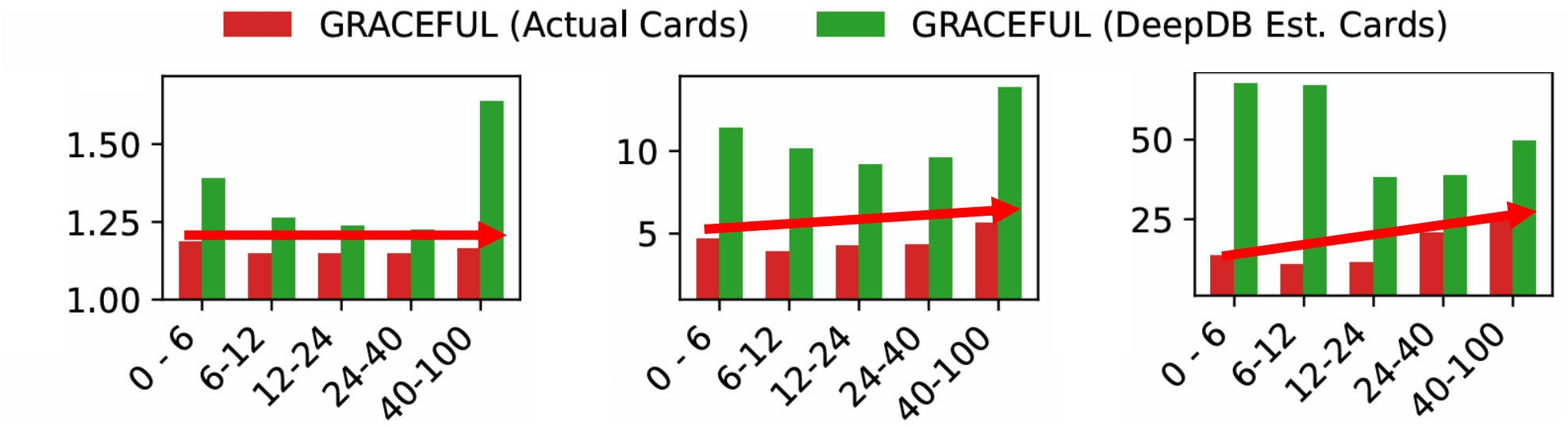
1. GNN-based Cost-Estimator For UDFs*
2. Transferable Representation for UDFs
3. Almost maximal End-to-End benefits for Pull-Up / Push-Down Optimization
4. Publishing UDF Benchmark and Source-Code

* that can generalize across UDFs, SQL workloads and datasets

Questions

Evaluation – Error with UDF Complexity

Graph Size (Number of COMPUTATION nodes)



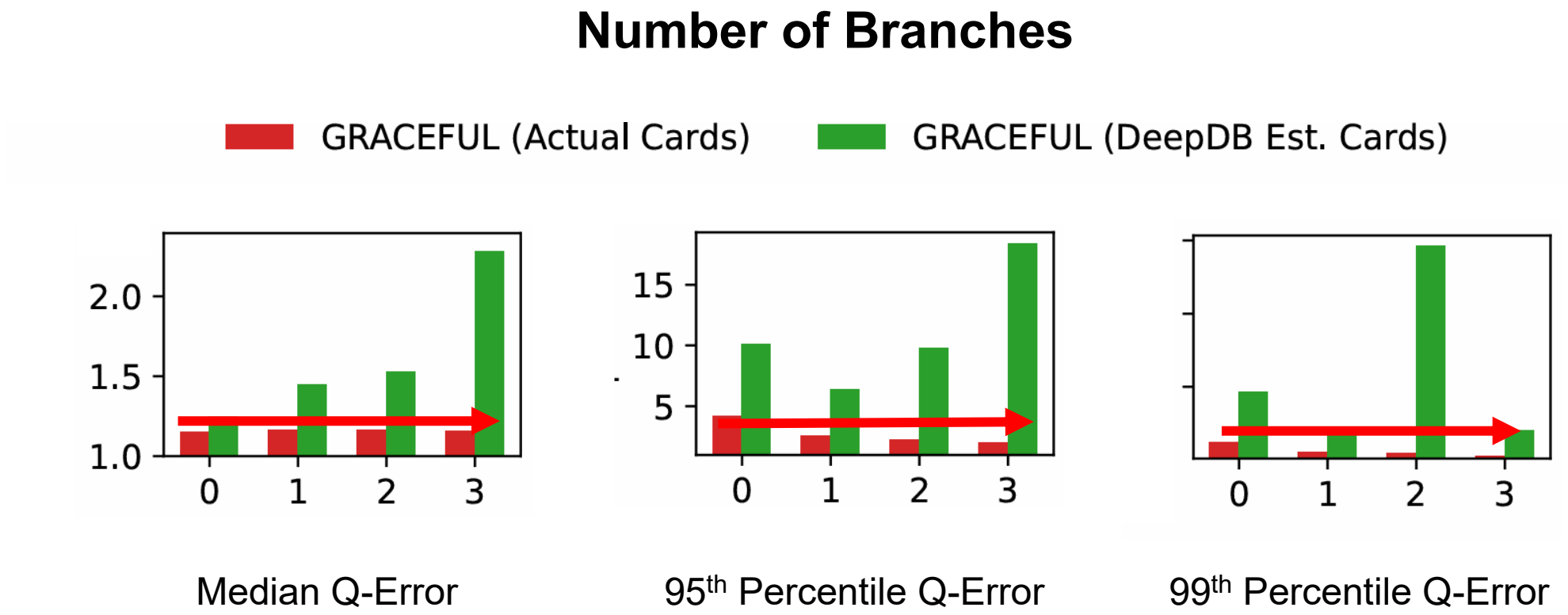
Median Q-Error

95th Percentile Q-Error

99th Percentile Q-Error

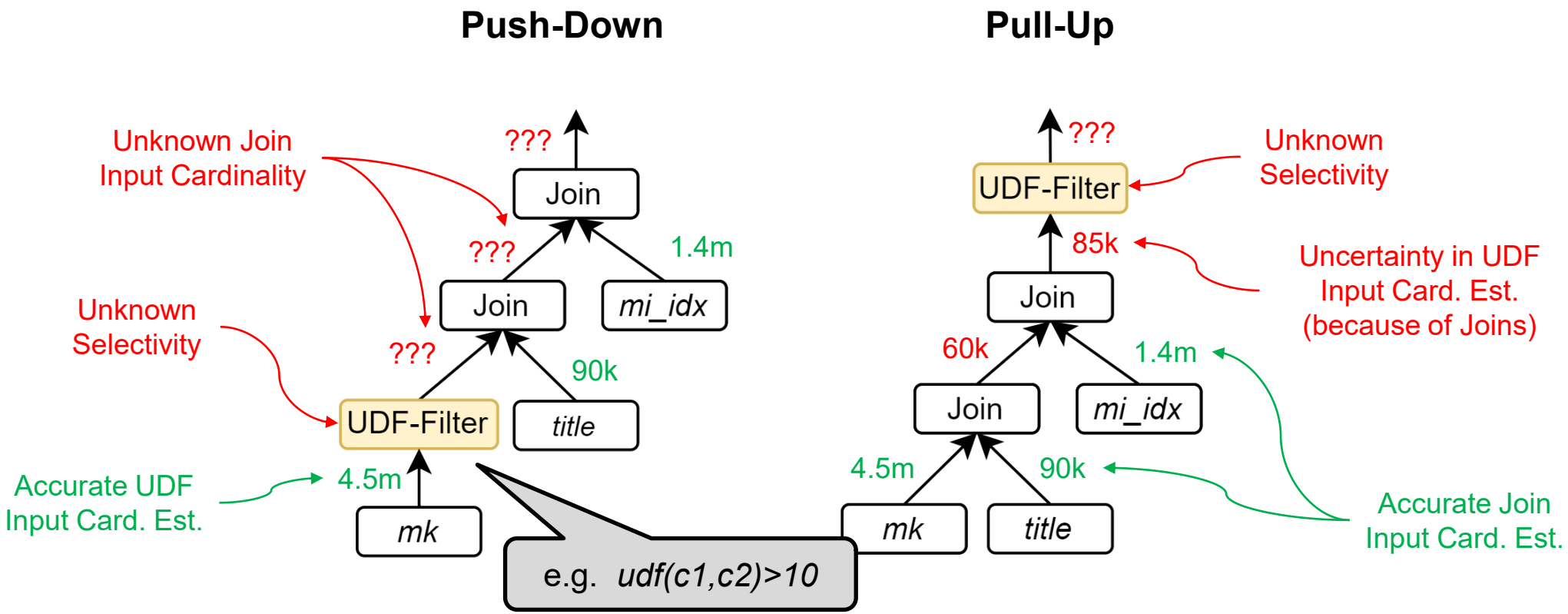
Scales with number of computations in UDF

Evaluation – Error with UDF Comlexity



Scales with number of Branches

Uncertainties in Push-Down vs. Pull-Up

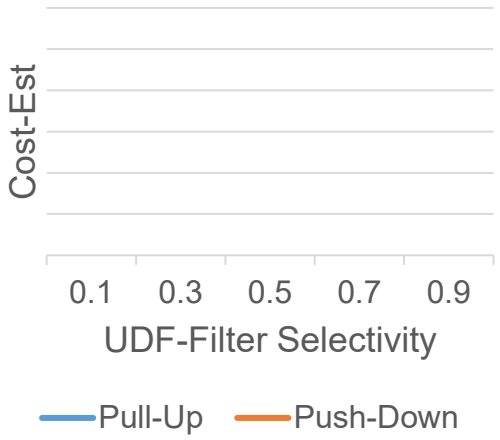


Higher Uncertainty in Join Cost

Higher Uncertainty in UDF Cost

Comparing Cost Distributions

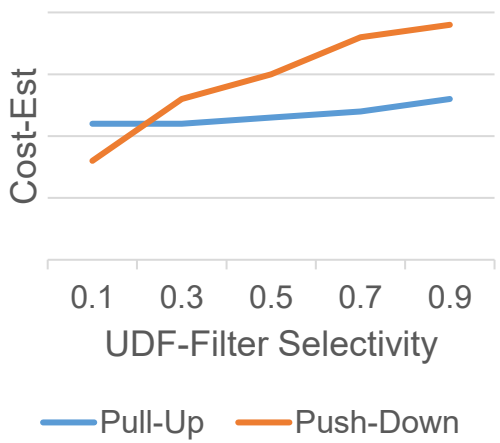
4 Strategies:



Never Pull

(Default in DBMS)

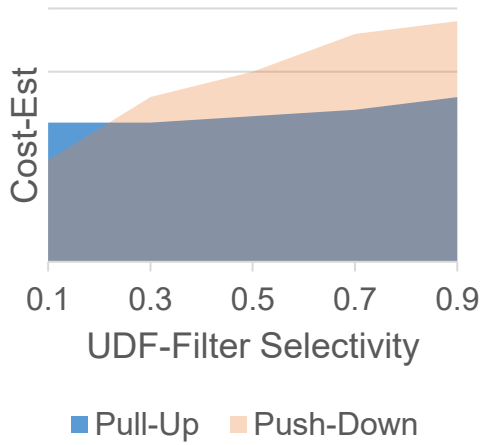
Push-Down



Conservative

Only Pull-Up if always beneficial

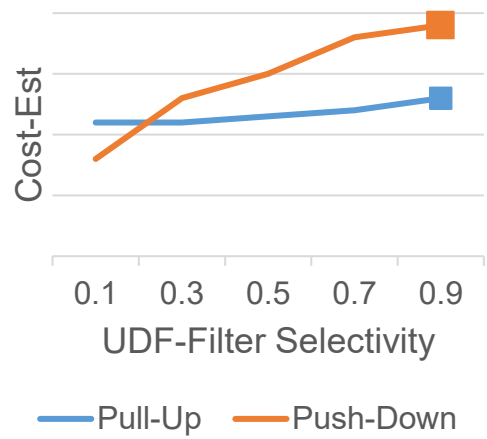
Push-Down



Area-Under-Curve

Select lower AuC

Pull-Up



Upper-Bound-Cardinality

Decide using Cost from $\mathrm{UDF{-}Filter\ Selectivity} = 1$

Pull-Up

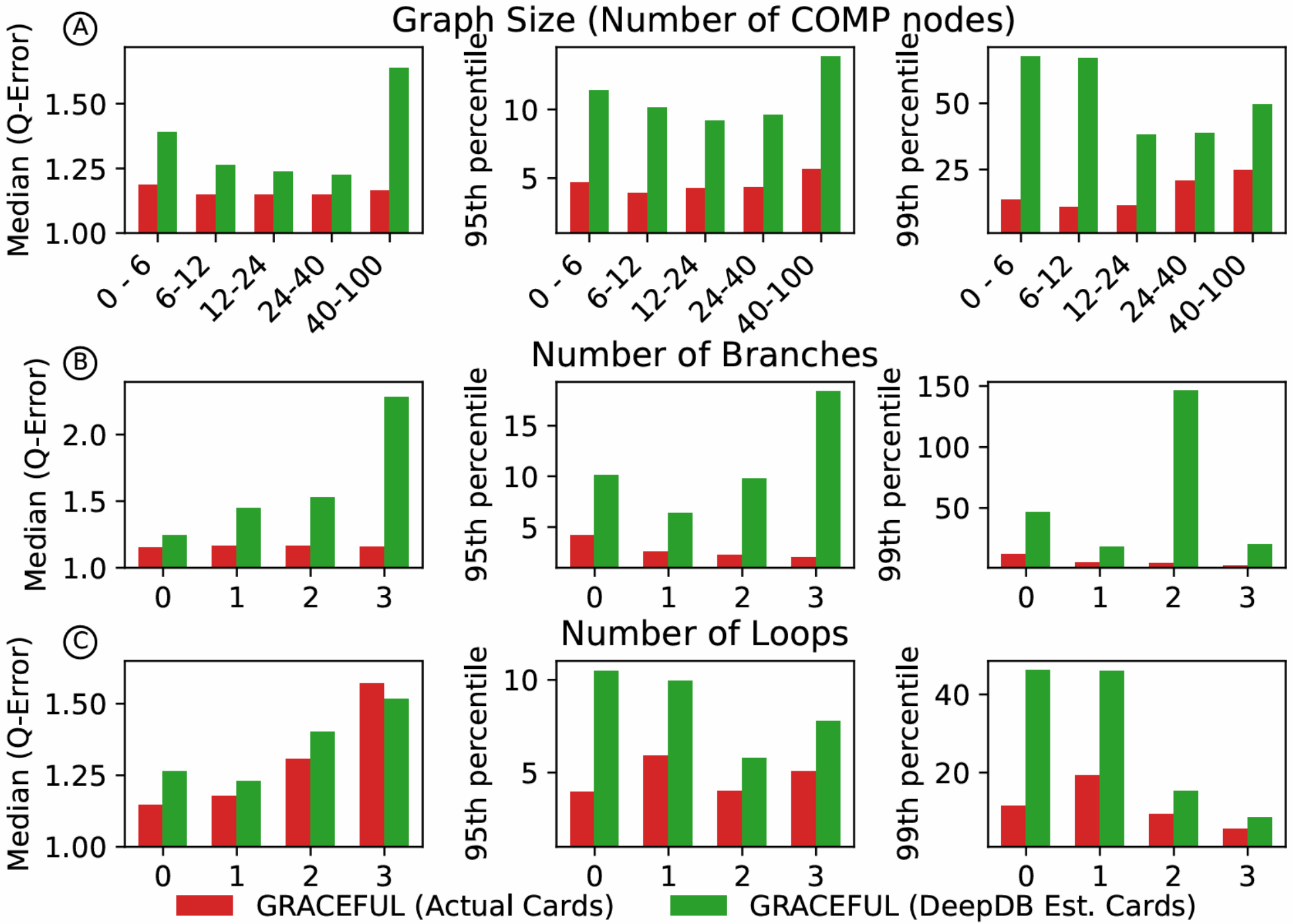
Evaluation

Further metrics in the paper

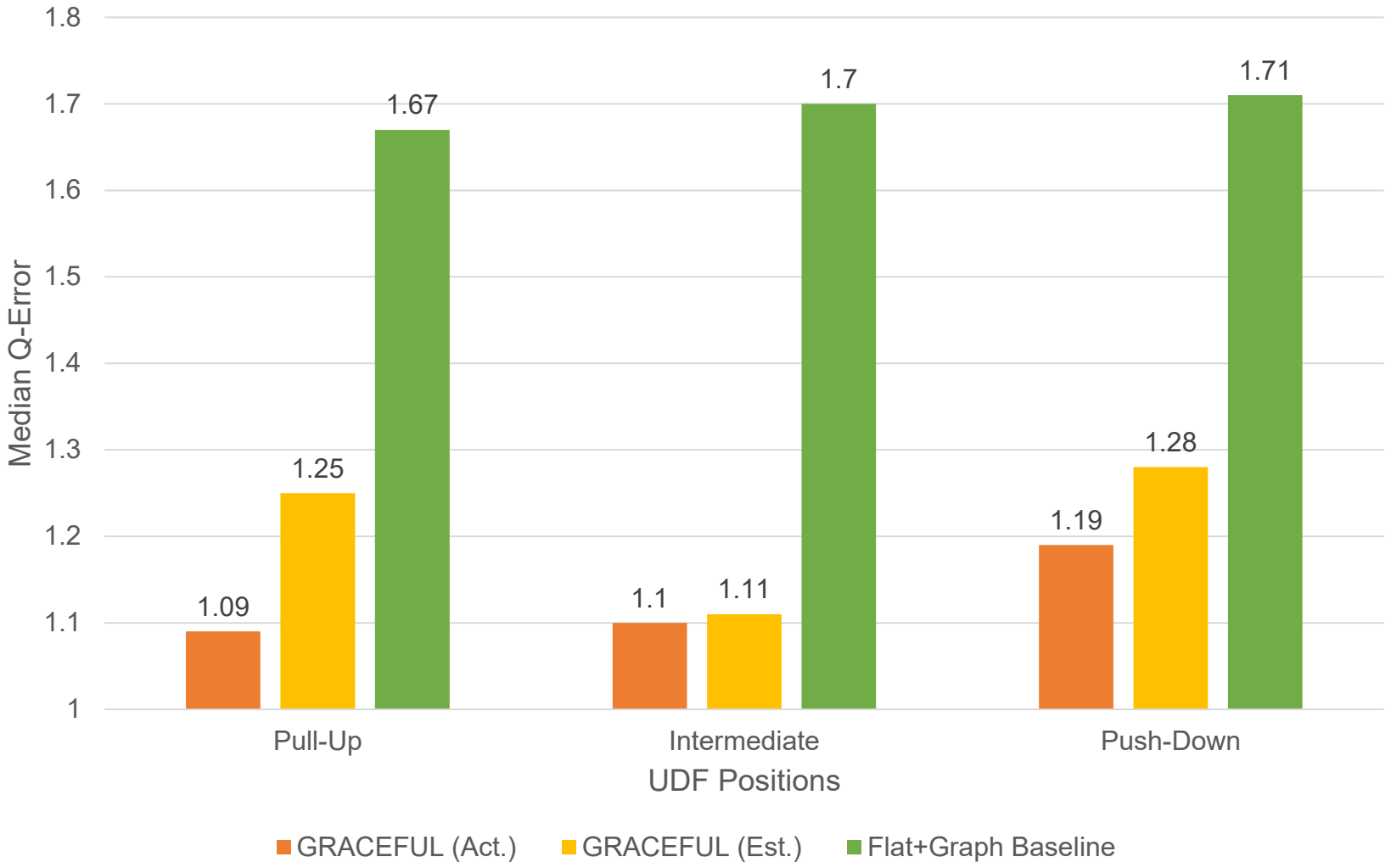
Selection Strategy	Card. Est. Method	Total Runtime (hrs)	Total Speedup	Median Speedup
Optimal	-	3.082	**1.643**	1.375
GRACEFUL (Cost)	Actual	3.217	**1.574**	1.370
GRACEFUL (Conservative)	DeepDB	3.460	**1.463**	1.331
GRACEFUL (AuC)	DeepDB	3.536	**1.432**	1.329
GRACEFUL (UBC)	DeepDB	3.595	**1.408**	1.316
No Pull-Up	-	5.063	1	1

1.57 – 1.40x Speedups (>1.5hrs)

Overhead of our Optimizer: 3-3.5% of workload runtime (unoptimized system)


Graph Size (Number of COMP nodes)
Number of Branches
Number of Loops
Median (Q-Error)
95th percentile
99th percentile
0 - 6
6-12
12-24
24-40
40-100
GRACEFUL (Actual Cards)
GRACEFUL (DeepDB Est. Cards)

Evaluation – Median Q-Error

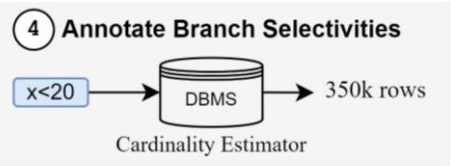


Q-Error:

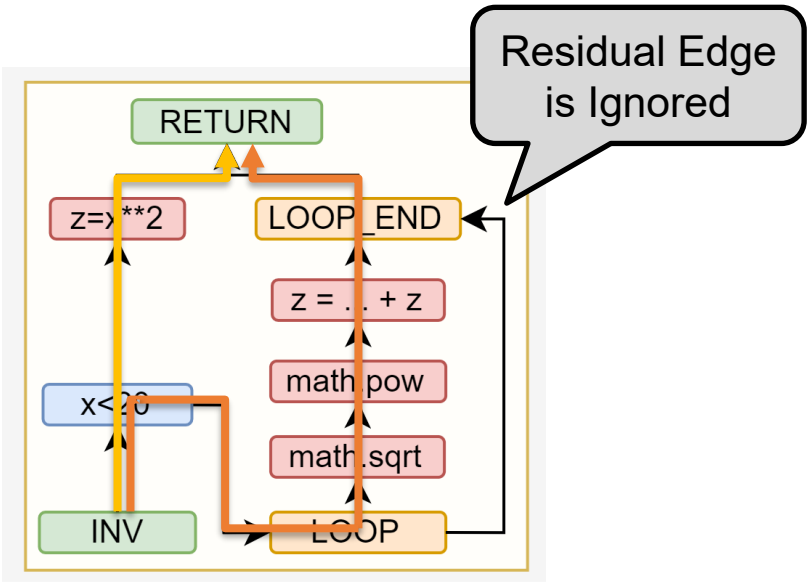
Relative Error Metric
(Lower is better, 1 is perfect)

Low Q-Error independent of positioning of the UDF

Annotate Branch Selectivities




```
SELECT COUNT( ∗) FROM title as t
WHERE func(t.year, t.id) > 5
   AND t.country = "GER";
```



Idea:

Leverage Database Statistics to estimate Branch Selectivities

Simple but powerful approach:

1. Extract all execution paths from UDF
2. Rewrite all conditions to SQL Query
3. Ask the DBMS Cardinality Estimator
4. Annotate Selectivities to nodes

Path 1: t.country = "GER" AND t.year < 20 → **1.5m tuples**

Path 2: t.country = "GER" AND t.year ≥ 20 → **30k tuples**

Training Data & Benchmark

To train & benchmark the model, we synthetically generated a benchmark (based on findings from Gupta et al.):

Number of Queries: 93.8k

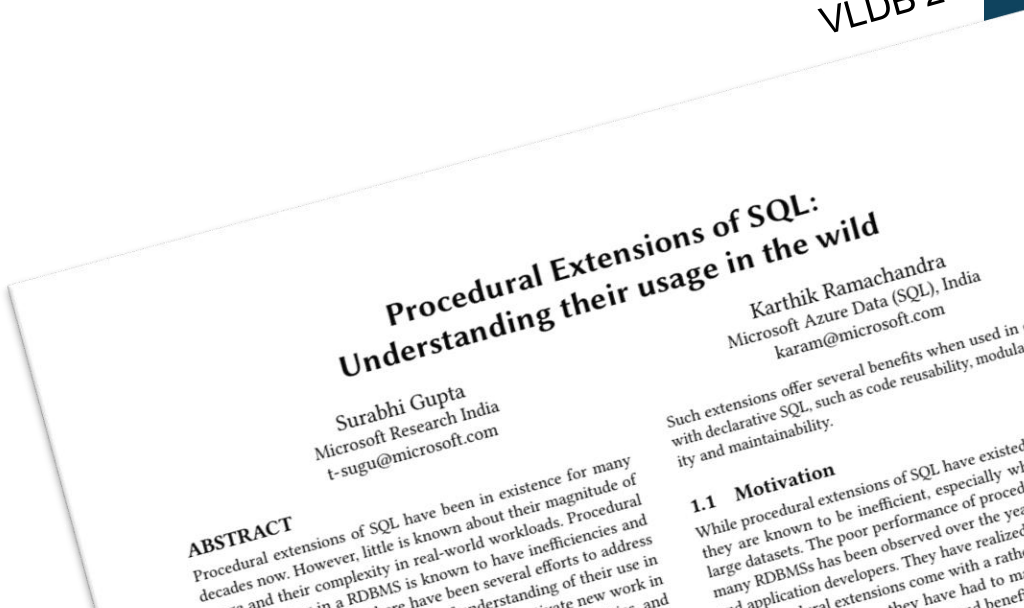
- 72k with UDFs in filters / 21k with UDFs in projection

Number of Databases: 20

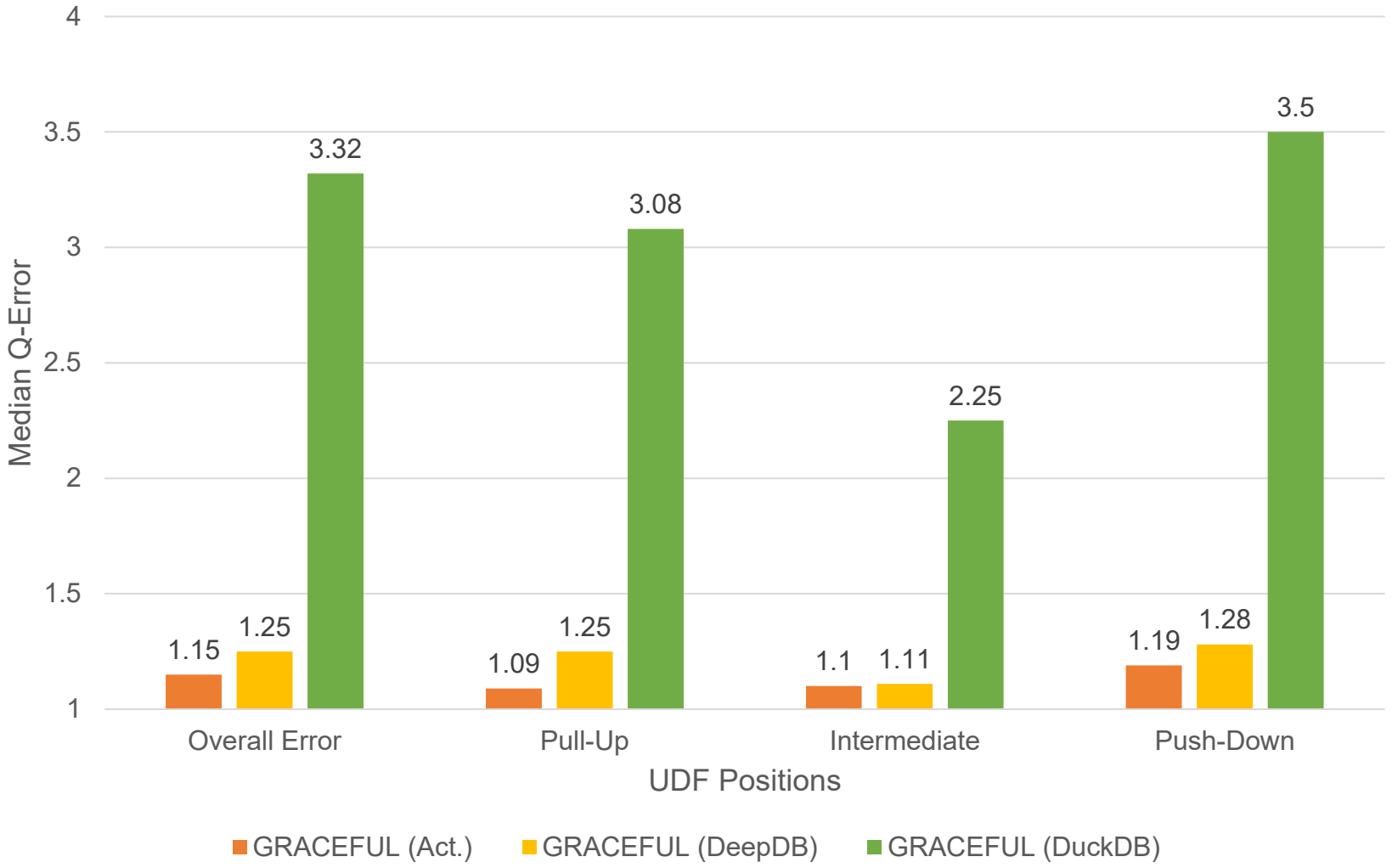
Query Complexity: 1-5 Joins, 0-21 Filters

UDF Complexity:

- Num branches: 0-3
- Num Loops: 0-3
- Num Arithmetic / String Ops: 10-150
- Supported Libraries: Math, Numpy



Evaluation – Median Q-Error



Q-Error:

Relative Error Metric
(Lower is better, 1 is perfect)

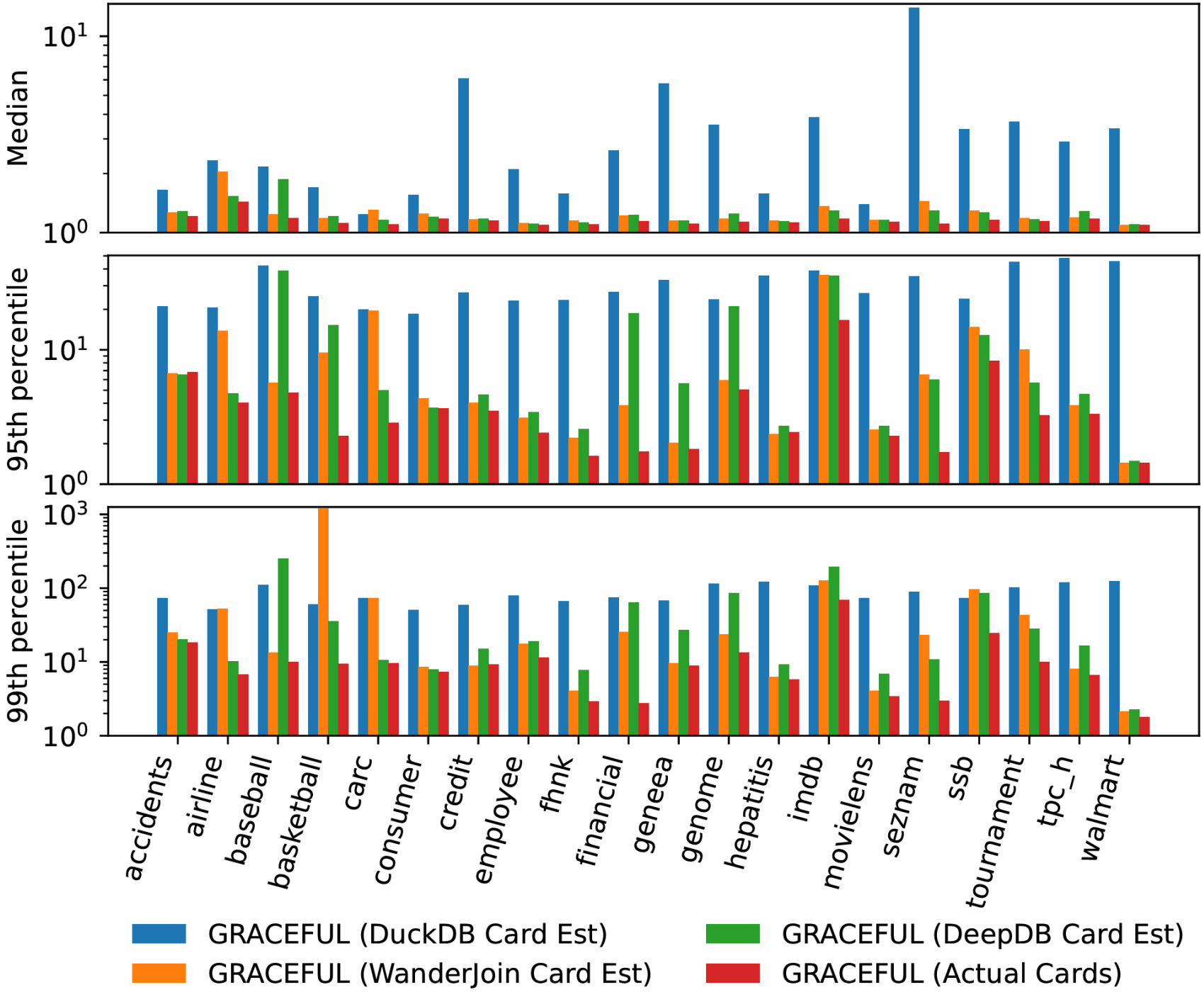
Card-Est Q-Errors:

	Med.	**95th**
Act	-	-
DeepDB	1.47	247.08
DuckDB	6.29	528.43

Low Q-Error independent of positioning of the UDF

Evaluation

Model	Card. Est. Method	Overall Error			Pull-Up			Intermediate Position			Push-Down			Card. Est. Error	
		Med.	95th	99th	Med.	95th	99th	Med.	95th	99th	Med.	95th	99th	Med.	95th
GRACEFUL	Actual	**1.15**	3.99	11.66	**1.09**	1.48	2.00	**1.10**	1.62	2.87	**1.19**	5.08	18.94	-	-
Flat+Graph	Actual	**1.71**	7.88	33.14	**1.67**	6.97	29.08	**1.70**	7.16	28.85	**1.71**	7.94	33.35	-	-
Graph+Graph	Actual	**2.61**	215.64	792.05	**2.17**	74.54	255.38	**2.65**	218.21	526.93	**2.72**	229.41	849.81	-	-
GRACEFUL	DeepDB [3]	**1.25**	10.08	45.17	**1.25**	10.49	460.99	**1.11**	1.76	2.77	**1.28**	11.19	44.15	**1.47**	247.08
GRACEFUL	WanderJoin [4]	**1.26**	7.89	88.46	**1.75**	14.07	31.52	**1.13**	1.71	2.60	**1.25**	7.23	84.87	**1.21**	309.38
GRACEFUL	DuckDB	**3.32**	30.14	84.70	**3.08**	40.42	132.48	**2.25**	24.89	177.52	**3.50**	28.76	80.30	**6.29**	528.43


Median
95th percentile
99th percentile
accidents
airline
baseball
basketball
carc
consumer
credit
employee
fhnk
financial
geneea
genome
hepatitis
imdb
movielens
seznam
ssb
tournament
tpc_h
walmart
GRACEFUL (DuckDB Card Est)
GRACEFUL (WanderJoin Card Est)
GRACEFUL (DeepDB Card Est)
GRACEFUL (Actual Cards)